МойОфис

Комплект Средств Разработки (SDK)

Руководство программиста

MYOFFICE DOCUMENT API (C++)

ООО «НОВЫЕ ОБЛАЧНЫЕ ТЕХНОЛОГИИ»

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ «МОЙОФИС КОМПЛЕКТ СРЕДСТВ РАЗРАБОТКИ (SDK)» MYOFFICE DOCUMENT APPLICATION PROGRAMMING INTERFACE (API). БИБЛИОТЕКА ДЛЯ ЯЗЫКА ПРОГРАММИРОВАНИЯ C++

РУКОВОДСТВО ПРОГРАММИСТА

2022.01

На 246 листах

Москва

2023

Все упомянутые в этом документе названия продуктов, логотипы, торговые марки и товарные знаки принадлежат их владельцам.

Товарные знаки «МойОфис» и «MyOffice» принадлежат ООО «НОВЫЕ ОБЛАЧНЫЕ ТЕХНОЛОГИИ».

Ни при каких обстоятельствах нельзя истолковывать любое содержимое настоящего документа как прямое или косвенное предоставление лицензии или права на использование товарных знаков, логотипов или знаков обслуживания, приведенных в нем. Любое несанкционированное использование этих товарных знаков, логотипов или знаков обслуживания без письменного разрешения их правообладателя строго запрещено.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

В настоящем документе используются следующие сокращения (см. таблицу 1):

Таблица 1. Сокращения и расшифровки

Сокращение	Расшифровка
ОС	Операционная система
MyOffice Document API	Программное обеспечение «МойОфис Комплект Средств Разработки (SDK). MyOffice Document API. Библиотека для языка программирования C++»
API	Application Programming Interface (программный интерфейс приложения)
IDE	Integrated Development Environment (интегрированная среда разработки)
SDK	Software Development Kit (комплект для разработки программного обеспечения)

1 Общие сведения

1.1 Назначение библиотеки

Библиотека MyOffice Document API для языка программирования C++ предназначена для использования в составе прикладных информационных систем или отдельных приложений под управлением ОС Microsoft Windows или Linux. Библиотека предназначена для решения задач по созданию и наполнению текстовых и табличных документов в пакетном режиме.

1.2 Библиотека MyOffice Document API для языка программирования C++

Библиотека MyOffice Document API для языка программирования C++ предоставляет возможность выполнения следующих операций:

1. Создание, открытие, сохранение изменений в электронных текстовых и табличных документах в следующих форматах:
   - текстовые и табличные документы, создаваемые с помощью Microsoft Office в формате OOXML, расширения файлов DOCX и XLSX;
   - текстовые и табличные документы, создаваемые с помощью LibreOffice в формате ODF, расширения файлов ODT и ODS;
   - текстовые и табличные документы, создаваемые с помощью МойОфис в формате ODF, расширения файлов XODT и XODS;
   - экспорт документов в формате PDF.
2. Изменение содержимого документов в пакетном режиме, в том числе:
   - добавление, удаление, изменение текста абзаца;
   - вставка, удаление, форматирование таблиц в текстовом документе;
   - вставка, удаление, переименование отдельных листов в табличном документе;
   - установка значения ячейки электронной таблицы и расчет формул;
   - оформление документа с использованием различных шрифтов и цветового оформления.
3. Поиск и замена фрагмента текста в документе.
4. Управление режимом рецензирования документа, отслеживание изменений в документе.
5. Управление закладками в текстовом документе.
6. Написание и запуск макрокоманд.

Для управления содержимым документа используется объектная модель, представляющая собой совокупность структур данных текстового или табличного документа.

1.3 Уровень подготовки пользователя

Пользователь MyOffice Document API должен иметь следующий опыт:

1. Разработка на языке C++ для ОС Microsoft Windows или Linux. Полный список поддерживаемых ОС приведен в документе «МойОфис Комплект Средств Разработки (SDK). MyOffice Document Application Programming Interface (API). Системные требования».
2. Работа со стандартными офисными приложениями.

1.4 Системные требования

Сборку приложения можно осуществить с помощью утилит командной строки. Предварительно необходимо убедиться, что используемая версия инструментария позволяет выполнить сборку 64-разрядного кода.

При сборке приложения для ОС Linux требуется наличие установленной библиотеки для сжатия данных zlib.

Полный перечень требований к программному и аппаратному обеспечению приведен в документе «МойОфис Комплект Средств Разработки (SDK). MyOffice Document Application Programming Interface (API). Системные требования».

2 Подготовка к работе

2.1 Список дистрибутивов

Дистрибутив MyOffice Document API поставляется в виде архивных файлов (см. таблицу 2).

Таблица 2 – Список дистрибутивов MyOffice Document API

ОС	Дистрибутив
Microsoft Windows	MyOffice_SDK_Document_API_Cpp_Win_2022.01_x64.zip
Linux	MyOffice_SDK_Document_API_Cpp_Linux_2022.01_x64.zip

2.2 Установка

Для установки MyOffice Document API необходимо извлечь содержимое архивного файла дистрибутива для соответствующей ОС (см. таблицу 2) в каталог установки MyOffice Document API.

После извлечения в каталоге установки MyOffice Document API будет создана папка **MyOfficeDocumentAPI**, содержащая следующие подкаталоги: **include**, **lib**, **share**.

Каталог **include** содержит заголовочные файлы MyOffice Document API, необходимые для сборки приложения на языке C++.

Каталог **lib** содержит:

- библиотеки, необходимые для сборки приложения, содержащего вызовы MyOffice Document API;
- динамические библиотеки, необходимые для запуска приложения, содержащего вызовы MyOffice Document API.

Каталог **share** содержит:

- ресурсы, необходимые для поддержи локализации;
- файлы для сборки приложения с помощью утилиты CMake;
- папку **examples** с примерами использования MyOffice Document API.

2.3 Сборка приложения для ОС Microsoft Windows

Сборка тестового приложения для ОС Microsoft Windows может быть осуществлена следующими способами:

- с использованием IDE;
- с помощью командной строки.

В папке **MyOfficeDocumentAPI\examples\BasicApplication** находится тестовый пример, который позволяет создать текстовый документ, добавить в него содержимое, сохранить документ с заданным именем и расширением:

```
#include <Core/Application.h>
#include <Document/Document.h>
#include <Document/DocumentType.h>
#include <Document/Position.h>
#include <Document/Range.h>
#include <Exceptions/Exceptions.h>
#include <iostream>
using namespace CO::API;

int main()
{
    try
    {
     // Создание экземпляра класса для управления параметрами и
     // объектами приложения
     Application application;

     // Создание нового текстового документа
     auto document = application.createDocument(Document::DocumentType::Text);

     // Работа с документом - вставка текста
     document.getRange().getBegin().insertText("Hello! This is an example!");

     // Сохранение документа
     auto outputFile = "./BasicExample.docx";
     document.saveAs(outputFile);
     std::cout << "Done: the '" << outputFile << "' file has been created." <<
std::endl;
     return 0;
    }

    // Обработка ошибок с диагностикой
    catch (const BaseError& e)
    {
        std::cerr << "FATAL ERROR: " << e.what() << std::endl;
    }
    catch (...)
    {
        std::cerr << "FATAL ERROR: Unknown internal error" << std::endl;
    }
    return 1;
};
```

2.3.1 Сборка приложения с использованием IDE

2.3.1.1 Настройка и сборка приложения в среде Microsoft Visual Studio

В данном разделе рассмотрен процесс настройки и сборки проекта в среде Microsoft Visual Studio с использованием библиотеки MyOffice Document API для языка C++.

Предварительно необходимо создать переменную окружения ОС Microsoft Windows с именем **MO_SDK** и присвоить ей в качестве значения строку, содержащую путь к папке **MyOfficeDocumentAPI** каталога установки MyOffice Document API.

Для продолжения настройки необходимо запустить Microsoft Visual Studio и создать новый проект, выбрав следующие настройки:

– тип проекта: консольное приложение C++;
– имя проекта: BasicApplication;
– папка расположения: **c:\Project**;
– имя решения: BasicApplication.

В окне редактора Microsoft Visual Studio необходимо заменить содержимое файла **BasicApplication.cpp** на содержимое файла тестового примера **main.cpp**, расположенного в папке **MyOfficeDocumentAPI\examples\BasicApplication** каталога установки MyOffice Document API.

Далее нужно настроить следующие свойства конфигурации проекта для активной конфигурации:

1. Указать каталог для поиска включаемых файлов, используя переменную окружения **MO_SDK** (см. Рисунок 1). Файлы заголовков для библиотеки MyOffice Document API расположены в папке **MyOfficeDocumentAPI\include** каталога установки MyOffice Document API.

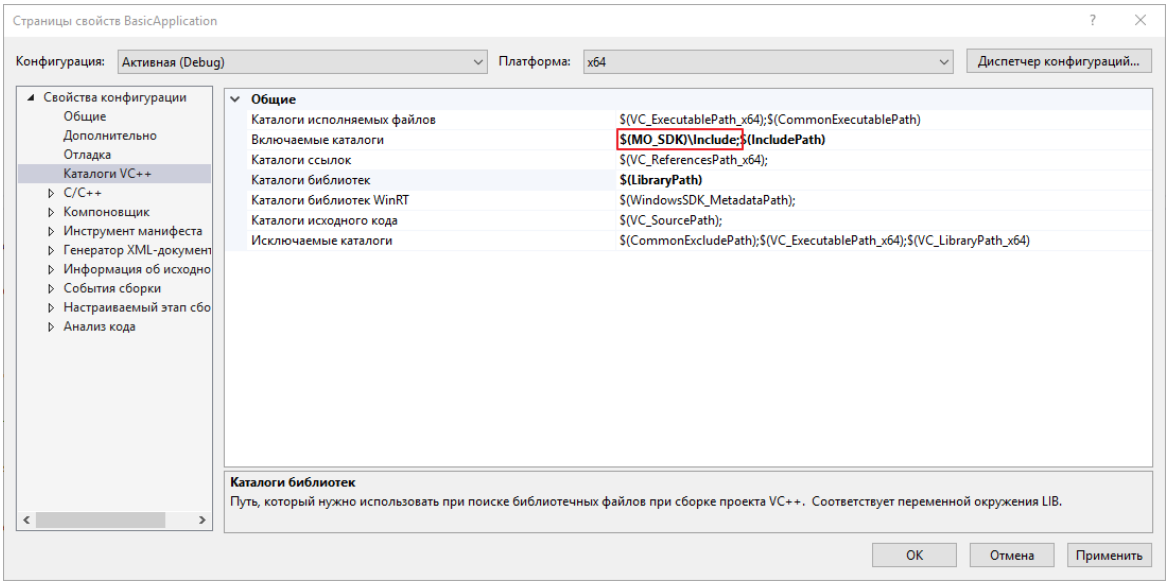


Рисунок 1 – Настройка каталогов включаемых файлов

2. Указать каталог для поиска файлов библиотек, используя переменную окружения **MO_SDK** (см. Рисунок 2). Файлы библиотек расположены в папке **MyOfficeDocumentAPI\lib** каталога установки MyOffice Document API.

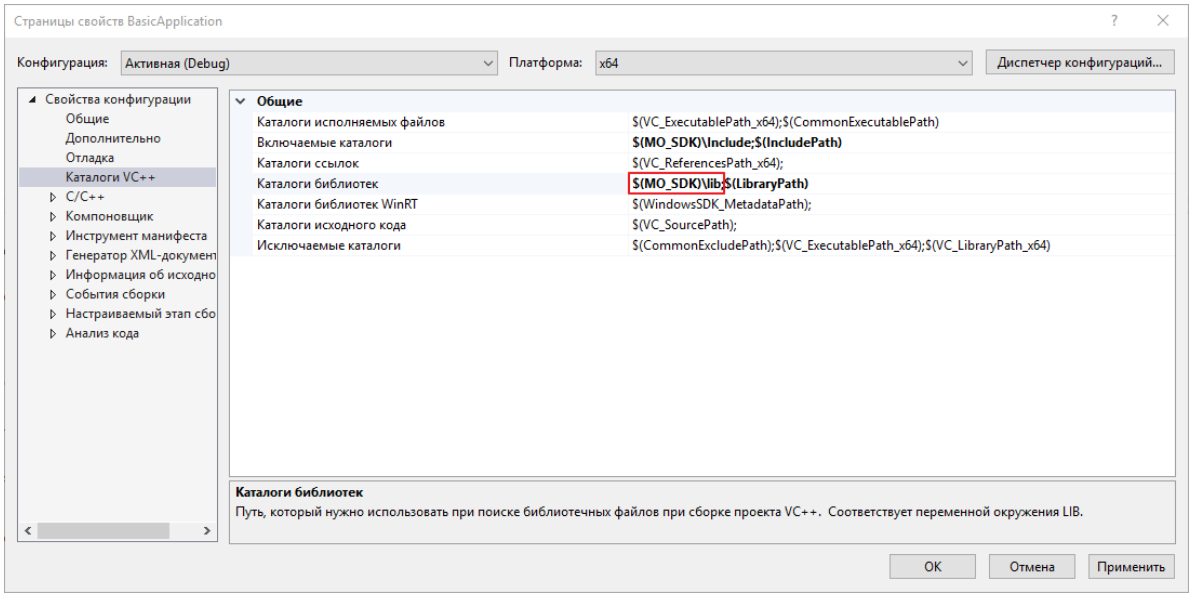


Рисунок 2 – Настройка каталогов библиотек

3. Добавить в настройки препроцессора следующие ключи (см. Рисунок 3):

```
BOOST_ALL_NO_LIB
BOOST_MPL_CFG_NO_PREPROCESSED_HEADERS
BOOST_MPL_LIMIT_LIST_SIZE=30
BOOST_MPL_LIMIT_VECTOR_SIZE=30
BOOST_SPIRIT_THREADSAFE
BOOST_STACKTRACE_GNU_SOURCE_NOT_REQUIRED
BOOST_SYSTEM_NO_DEPRECATED
BOOST_USE_WINDOWS_H
CEREAL_DLL_EXPORT=/**/
CO_EXPORT_SYMBOLS=1
NOMINMAX=1
UCLN_NO_AUTO_CLEANUP=0
U_DISABLE_RENAMING=1
U_ENABLE_DYLOAD=0
U_STATIC_IMPLEMENTATION
U_USING_ICU_NAMESPACE=0
WIN32_LEAN_AND_MEAN
XERCES_STATIC_LIBRARY
XSD_CXX11
ZIP_STATIC
WEBSOCKETPP_CPP11_STL_
WINDOWS
WIN32
```

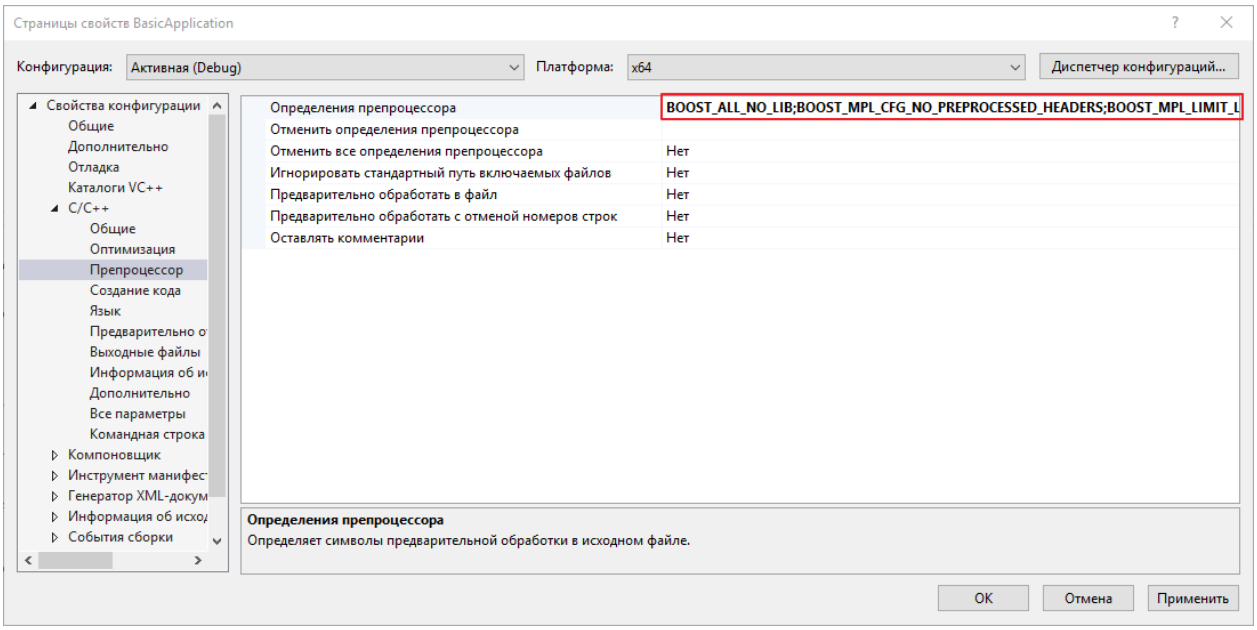


Рисунок 3 – Настройка определений препроцессора

4. Указать в конфигурации компоновщика библиотеку **MyOfficeDocumentAPI.lib** для конфигурации **Release** или **MyOfficeDocumentAPId.lib** для конфигурации **Debug** в качестве дополнительной зависимости (см. Рисунок 4). Данные библиотеки расположены в папке **MyOfficeDocumentAPI\lib** каталога установки MyOffice Document API.

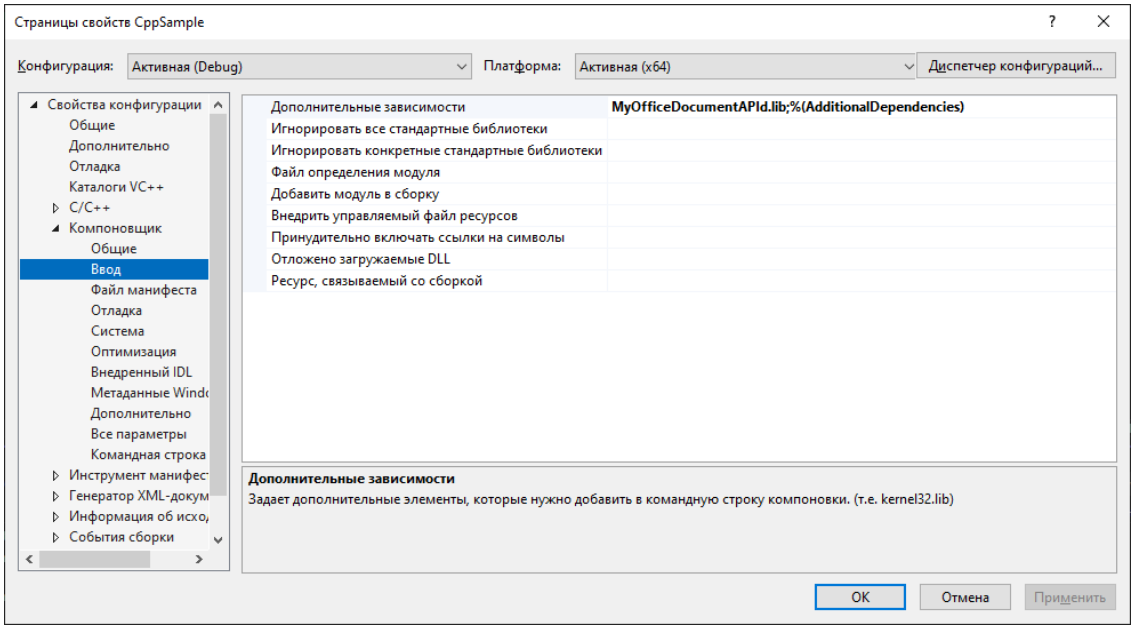


Рисунок 4 – Настройка дополнительных зависимостей

Для сборки приложения в командном меню Microsoft Visual Studio необходимо выбрать пункт **Сборка > Собрать решение**.

2.3.1.2 Проверка работоспособности

Для проверки работоспособности MyOffice Document API необходимо произвести сборку тестового примера в Microsoft Visual Studio в соответствии с разделом Настройка и сборка приложения, а затем осуществить следующие действия:

1. Скопировать файл динамической библиотеки **MyOfficeDocumentAPId.dll** из папки **MyOfficeDocumentAPI\lib** каталога установки MyOffice Document API в папку **c:\Project\BasicApplication\x64\Debug**, либо, в случае релизной сборки, скопировать **MyOfficeDocumentAPI.dll** в папку **....\x64\Release**.
2. Скопировать папку **Resourses**, содержащую ресурсы приложения, из папки **MyOfficeDocumentAPI\share** каталога установки **MyOffice Document API** в папку **c:\Project\BasicApplication**;
3. Запустить собранное приложение, выбрав в командном меню пункт **Отладка > Запуск без отладки**.

В результате выполнения приложения в папке **c:\Project\BasicApplication** будет создан файл **BasicExample.docx**, а в окне консоли отладки Microsoft Visual Studio отобразится сообщение выполненного приложения, а также код его завершения.

MyOffice Document API считается работоспособным, если приложение выполнено успешно (код завершения равен нулю).

2.3.2 Сборка приложения из командной строки

Для сборки приложения из командной строки необходимы следующие утилиты:

– **cmake** – утилита для сборки программы из исходного кода;
– **make** – утилита для компиляции исходного кода в объектные файлы и последующей компоновки в исполняемые файлы.

2.3.2.1 Сценарий сборки

Для использования утилиты CMake необходим файл сценария сборки **CMakeLists.txt**, в котором описаны правила и цели сборки.

Пример файла **CMakeLists.txt** для сборки приложения тестового примера (см. раздел Сборка приложения для MS Windows):

```
# Проверка версии CMake, требуется не ниже 3.1
make_minimum_required(VERSION 3.1)
project(BasicApplication)

# Проверка указан ли путь к каталогу установки MyOffice Document API, если не указан -
```



```
ошибка
if(NOT DEFINED SDK_PATH)
    message(FATAL_ERROR "Please, specify the path to SDK with -DSDK_PATH=path\to\sdk")
endif()

# Установка флага компилятора, т. к. MyOffice Document API требует поддержки C++11
set(CMAKE_CXX_STANDARD 11)

# Поиск пути к библиотеке MyOffice Document API
find_package(MyOfficeDocumentAPI REQUIRED NO_MODULE PATHS "${SDK_PATH}" NO_DEFAULT_PATH NO_CMAKE_FIND_ROOT_PATH)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${CO_CORE_COMPILE_FLAGS_STRING}")

# Сборка выполняемого файла из файла исходного кода
add_executable(BasicApplication Main.cpp)

# Указание библиотеки MyOffice Document API для связи в процессе сборки
target_link_libraries(BasicApplication CO::Core)
```

2.3.2.2 Сборка приложения

Далее будет использован тестовый пример (см. раздел Сборка приложения для MS Windows) из файла **Main.cpp**, расположенного в папке **MyOfficeDocumentAPI\examples\BasicApplication** каталога установки MyOffice Document API.

Для сборки приложения из исходного кода тестового примера необходимо выполнить следующие действия:

1. В корневом каталоге диска **C:** создать папку **BasicApp** для размещения файла исходного кода.
2. Скопировать в созданную папку **BasicApp** файлы **CMakeLists.txt** и **Main.cpp** из папки **MyOfficeDocumentAPI\examples\BasicApplication** каталога установки MyOffice Document API.
3. Перейти в каталог **C:\BasicApp.**
4. Проверить наличие файла сценария **CMakeLists.txt** в текущей папке, при его отсутствии создать его на основе примера, приведенного в разделе Сценарий сборки.

5. Выполнить команду:

```
cmake . -DSDK_PATH=path\to\sdk
```

Где **path\to\sdk** – путь к папке **MyOfficeDocumentAPI** каталога установки.

6. Выполнть команду:

```
cmake --build .
```

Исполняемый файл **BasicApplication.exe** будет создан в папке **C:\BasicApp\Debug**.

2.3.2.3 Проверка работоспособности

Проверка работоспособности MyOffice Document API производится с использованием приложения, сборка которого описана в разделе Сборка приложения.

Перед проверкой необходимо скопировать в папку с приложением следующие объекты:

1. Файл динамической библиотеки **MyOfficeDocumentAPId.dll** для конфигурации **Debug** или **MyOfficeDocumentAPI.dll** для конфигурации **Release** из папки **MyOfficeDocumentAPI\lib** каталога установки MyOffice Document API.
2. Папку **Resourse**, содержащую ресурсы приложения, из папки **MyOfficeDocumentAPI\share** каталога установки MyOffice Document API.

Для проверки работоспособности MyOffice Document API необходимо запустить приложение.

MyOffice Document API считается работоспособным, если приложение выполнено успешно и в папке с приложением создан файл **BasicExample.docx**.

2.3.3 Распространение разработанных приложений

Для распространения разработанных приложений, использующих вызовы MyOffice Document API, необходимо обеспечить наличие следующих объектов в каталоге с распространяемым приложением:

1. Папка **Resources**, содержащая ресурсы приложения (скопировать папку **MyOfficeDocumentAPI\share\Resources** каталога установки MyOffice Document API).
2. Файл динамической библиотеки **MyOfficeDocumentAPId.dll** для конфигурации **Debug** или **MyOfficeDocumentAPI.dll** для конфигурации **Release** (скопировать из папки **MyOfficeDocumentAPI\lib** каталога установки MyOffice Document API).

2.4 Сборка приложения для ОС Linux

Сборка приложения для ОС Linux осуществляется с помощью командной строки.

Для сборки приложения из командной строки необходимы следующие утилиты:

- **cmake** – утилита, предназначенная для автоматизации сборки программы из исходного кода;
- **make** – утилита для компиляции исходного кода в объектные файлы и последующей компоновки в исполняемые файлы;
- **gcc**-совместимый компилятор (должен иметь 64-битную поддержку).

Для выполнения сборки приложения для ОС Linux требуется наличие установленной библиотеки для сжатия данных **zlib**.

2.4.1 Сценарий сборки

Для использования утилиты CMake необходим файл сценария сборки **CMakeLists.txt**, в котором описаны правила и цели сборки.

Пример файла **CMakeLists.txt** для сборки тестового примера:

```
# Проверка версии CMake, требуется не ниже 3.1
make_minimum_required(VERSION 3.1)
project(BasicApplication)

# Проверка указания пути к каталогу установки MyOffice Document API
if(NOT DEFINED SDK_PATH)
    message(FATAL_ERROR "Please, specify the path to SDK with -DSDK_PATH=path/to/sdk")
endif()

# Установка флага компилятора, т. к. MyOffice Document API требует поддержки C++11
set(CMAKE_CXX_STANDARD 11)

# Поиск пути к библиотеке MyOffice Document API
find_package(MyOfficeDocumentAPI REQUIRED NO_MODULE PATHS "${SDK_PATH}"
NO_DEFAULT_PATH NO_CMAKE_FIND_ROOT_PATH)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${CO_CORE_COMPILE_FLAGS_STRING}")

# Сборка выполняемого файла из файла исходного кода
add_executable(BasicApplication Main.cpp)
```

```
 # Указание библиотеки MyOffice Document API для связи в процессе сборки
target_link_libraries(BasicApplication CO::Core)
```

2.4.2 Сборка приложения

В качестве примера для сборки приложения будет использован файл **main.cpp**, расположенный в папке **MyOfficeDocumentAPI/examples/BasicApplication** каталога установки MyOffice Document API.

Для сборки тестового примера необходимо выполнить следующие действия:

1. Перейти в папку **MyOfficeDocumentAPI/examples/BasicApplication** каталога установки MyOffice Document API.
2. Выполнить команду:

```
cmake CMakeLists.txt -DSDK_PATH=path/to/sdk
```

   Где **path/to/sdk** – это путь к папке **MyOfficeDocumentAPI** каталога установки MyOffice Document API.

3. Выполнить команду:

```
cmake --build .
```

Исполняемый файл **BasicApplication** будет создан в папке **BasicApplication**.

2.4.3 Проверка работоспособности

Для проверки работоспособности MyOffice Document API необходимо запустить исполняемый файл, сборка которого описана в разделе Сборка приложения.

MyOffice Document API считается работоспособным, если исполняемый файл выполнен успешно и в папке запуска создан файл **BasicExample.docx**.

2.4.4 Распространение разработанных приложений

Для распространения приложения необходимы следующие файлы:

1. Исполняемый файл.
2. В каталоге с исполняемым файлом находится папка **Resources** (требуется скопировать папку **MyOfficeDocumentAPI/share/Resources** каталога установки MyOffice Document API).
3. В системном каталоге **/lib** находятся следующие файлы: **libMyOfficeDocumentAPI.so**, **libcrypto.so**, **libsbb.so**, **libssl.so** (требуется скопировать из папки **MyOfficeDocumentAPI/lib** каталога установки MyOffice Document API).

3 Объектная модель МойОфис SDK

МойОфис SDK предоставляет разработчику возможности для управления содержимым текстового и табличного документа.

Библиотека позволяет работать с пользовательскими документами различных форматов, однако, внутренняя модель документа представлена в формате ODF (Open Document Format, открытый формат документов для офисных приложений), который принят в качестве ГОСТ (Р ИСО/МЭК 26300-2010). Описание внутреннего формата ODF размещено на ресурсе сообщества OASIS (*Organization for the Advancement of Structured Information Standards*).

В данном документе описана объектная модель API (классы, коллекции, методы доступа) для доступа к компонентам внутренней модели документа.

Для этого используются классы, расположенные в двух основных пространствах имен (namespaces): `CO:API` и `CO:API:Document` (см. Рисунок 5). `CO:API` содержит основной класс `CO:API:Application`, который служит для создания и открытия документа. Помимо этого, `CO:API` содержит несколько вспомогательных классов. Пространство имен `CO:API:Document` содержит классы и функции для представления документа и всех его составляющих, которые поддерживает МойОфис: абзацы, таблицы, ячейки, рисунки, колонтитулы и т.д.

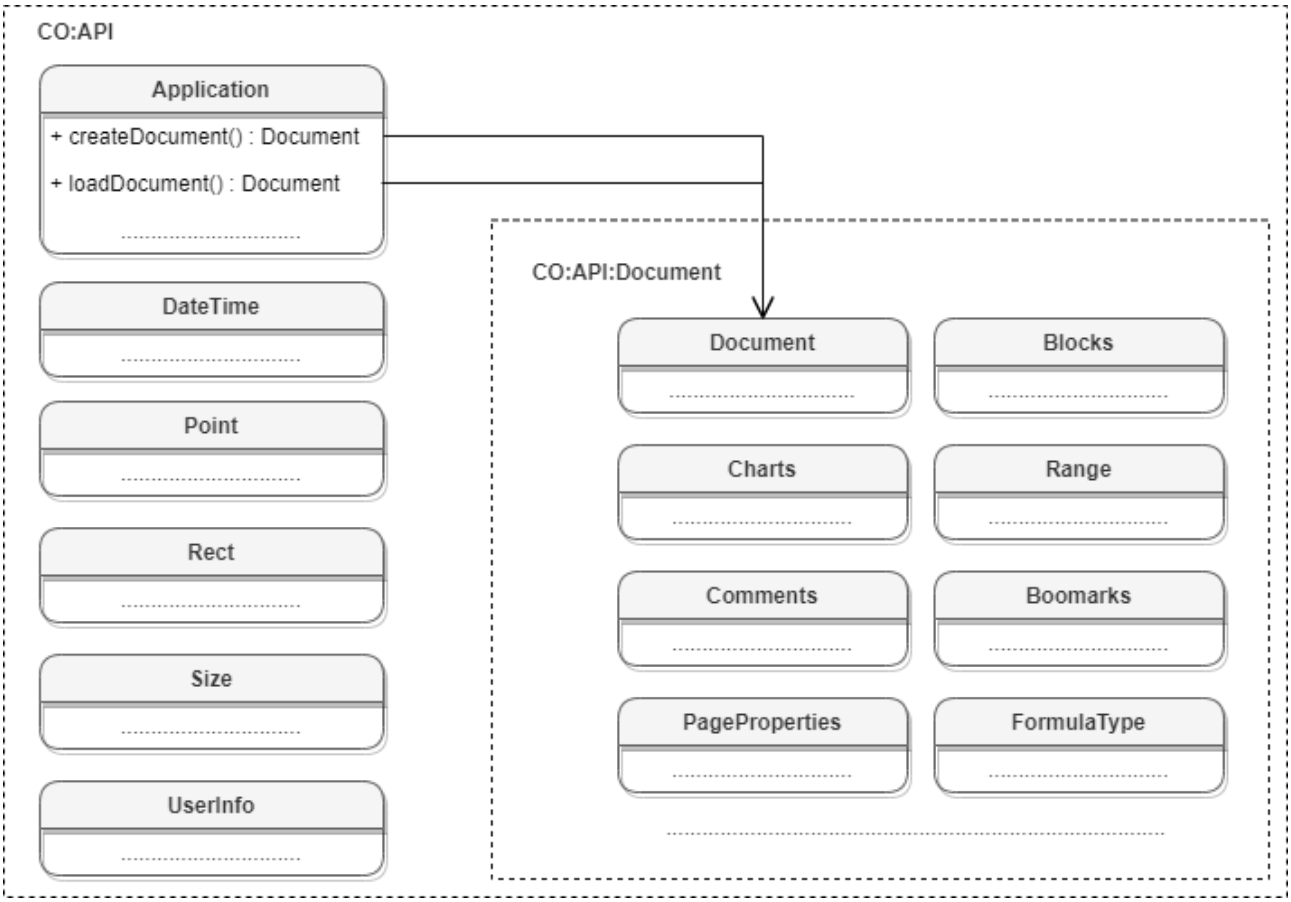


Рисунок 5 – Объектная модель МойОфис SDK.

4 Работа с документами

4.1 Работа с текстовым документом

4.1.1 Создание и открытие текстового документа

Метод Application::createDocument создает документ. В качестве параметра используются DocumentType или DocumentSettings.

Примеры создания текстового документа:

```
auto document = application.createDocument(DocumentType::Text);
```

```
DocumentSettings documentSettings = DocumentSettings();
documentSettings.documentType = DocumentType::Workbook;
auto document = application.createDocument(documentSettings);
```

Метод Application::loadDocument загружает документ. В качестве параметра используется путь к документу. Дополнительно может быть использован параметр LoadDocumentSettings.

Примеры загрузки текстового документа:

```
auto document = application.loadDocument("test.docx");
```

```
auto documentSettings = DocumentSettings();
documentSettings.documentType = DocumentType::Text;
auto loadSettings = LoadDocumentSettings();
loadSettings.commonDocumentSettings = documentSettings;
auto document = application.loadDocument("test.docx", loadSettings);
```

4.1.2 Сохранение и экспорт текстового документа

Метод Document::saveAs сохраняет документ по указанному пути.

Примеры сохранения текстового документа:

```
document.saveAs(filePath);
```

```
SaveDocumentSettings saveDocumentSettings = new SaveDocumentSettings();

saveDocumentSettings.documentFormat = DocumentFormat.OXML;
saveDocumentSettings.documentType = DocumentType.Text;
saveDocumentSettings.documentPassword = "password";
saveDocumentSettings.isTemplate = false;
```

```
saveDocumentSettings.dsvSettings = new DSVSettings();
saveDocumentSettings.dsvSettings.autofit = true;
saveDocumentSettings.dsvSettings.startBlockIndex = 0;
saveDocumentSettings.dsvSettings.lastBlockIndex = 10;

document.saveAs(filePath, saveDocumentSettings);
```

Метод Document::exportAs экспортирует документ в файл по указанному пути с заданным форматом типа ExportFormat.

В настоящее время поддерживается только операция экспорта документа в формат PDF.

Примеры экспорта текстового документа:

```
document.exportAs(filePath, ExportFormat.PDFA1);
```

```
TextExportSettings textExportSettings = new TextExportSettings();
textExportSettings.pageNumbers = new PageNumbers(PageParity.Even);
document.exportAs(filePath, ExportFormat.PDFA1, textExportSettings);
```

4.1.3 Разделы (секции) документа

На рисунке 6 изображена объектная модель классов, относящихся к работе с секциями текстового документа.

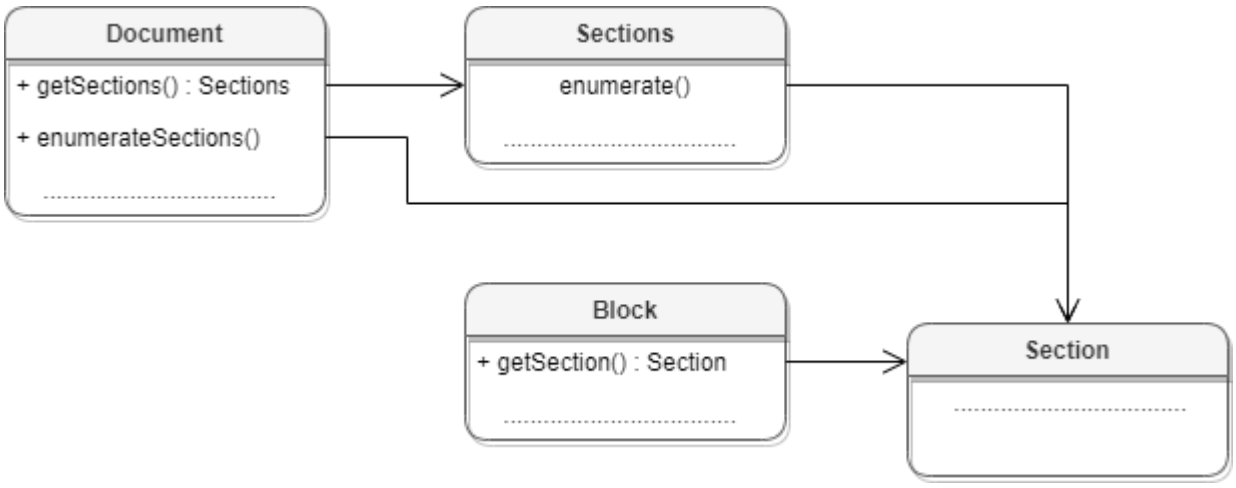


Рисунок 6 – Объектная модель классов для работы с секциями

Секция в текстовом документе - это раздел, который содержит страницы с одинаковыми параметрами, а также одинаковыми верхними и нижними колонтитулами.

Доступ к секциям текстового документа может быть осуществлен одним из следующих способов:

– получение объекта Sections с помощью вызова Document::getSections();
– перечисление всех доступных секций Section с помощью вызова Document::getSectionsEnumerator();
– получение секции Section вызовом метода Block::getSection() для блока, который входит в секцию.

Примеры:

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    Section section = enumerator->getCurrent();
    std::printf("%d", section.getPageProperties().width);
    enumerator->goToNext();
}
```

```
std::shared_ptr<Enumerator<Section>> enumerator =
document.getSectionsEnumerator();
while (enumerator->isValid()) {
    Section section = enumerator->getCurrent();
    std::printf("%d", section.getPageProperties().width);
    enumerator->goToNext();
}
```

```
boost::optional<Block> blockOpt = document.getBlocks().getBlock(0);
if (blockOpt.has_value()) {
    Section section = blockOpt.get().getSection();
    std::printf("%d", pageProperties.width);
}
```

4.1.4 Работа со встроенными объектами

Редакторы текста и таблиц МойОфис поддерживают несколько типов графических объектов со схожим поведением: изображения (Image) и фигуры (Shape), которые являются разновидностью фигур.

Объектная модель документа в части управления изображениями развивается и дополняется возможностями. Доступны следующие операции:

– вставка изображений в текстовый документ;

– перечисление графических объектов, находящихся в текстовом документе, определение их типа и геометрических размеров;
– перемещение графических объектов, находящихся в текстовом документе, изменение их размеров и масштаба;
– вставка изображений в текстовый документ.

Доступ ко встроенным объектам документа осуществляется посредством использования метода Range::getInlineObjects().

Пример:

```
InlineObjects inlineObjects = document.getRange().getInlineObjects();
```

4.1.4.1 Вставка изображения

Для вставки изображения используется метод Position::insertImage().

Вставка изображения в текстовый документ

```
Range range = document.getRange();
range.getBegin().insertImage("C://Tmp/123.jpg", Size<float>(50.0, 50.0));
```

4.1.4.2 Перечисление встроенных объектов

Перечисление графических объектов в текстовом документе.

```
InlineObjects inlineObjects = document.getRange().getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    boost::optional<Image> imageOpt = inlineObject.toImage();
    if (imageOpt.has_value()) {
        std::printf("Image");
    } else {
        std::printf("Shape");
    }
    enumerator->goToNext();
}
```

Перечисление изображений в текстовом документе.

```
Images images = document.getRange().getImages();
std::shared_ptr<Enumerator<Image>> enumerator = images.getEnumerator();
while (enumerator->isValid()) {
```

```
    Image image = enumerator->getCurrent();
    std::printf("Image");
    enumerator->goToNext();
}
```

4.1.4.3 Определение типа встроенных объектов

Для определения типа графического объекта (Image/Shape) может быть использован метод InlineObject::toImage(). В случае, если объект является изображением, метод вернет ненулевой объект.

```
Range range = document.getRange();
InlineObjects inlineObjects = range.getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    boost::optional<Image> imageOpt = inlineObject.toImage();
    if (imageOpt.has_value()) {
        // Image
        Image image = imageOpt.get();
    } else {
        // Not an image
    }
    enumerator->goToNext();
}
```

4.1.4.4 Изменение параметров встроенного объекта

Размеры графического объекта могут быть получены из объекта Frame, который может быть получен посредством использованием метода InlineObject::getFrame().

```
Range range = document.getRange();
InlineObjects inlineObjects = range.getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    Frame frame = inlineObject.getFrame();
    boost::optional<Size<float>> dimensionsOpt = frame.getDimensions();
    ..........................
```

```
    enumerator->goToNext();
}
```

Помимо этого, можно задавать такие параметры встроенных объектов как размер, позиция и способ обтекания текстом.

```
// Позиция встроенного объекта не может быть задана,
// если стиль переноса текста - inline.
// Сначала его следует изменить на тип, отличный от inline.
Frame frame = inlineObject.getFrame();
if (wrapType == TextWrapType::Inline) {
    frame.setWrapType(TextWrapType::TopAndBottom);
}
```

Используя классы HorizontalTextAnchoredPosition, VerticalTextAnchoredPosition, можно задать положение встроенных объектов в текстовом документе с учетом относительного смещения.

```
Frame frame = inlineObject.getFrame();
TextAnchoredPosition position = TextAnchoredPosition();
position.horizontal =
boost::optional<HorizontalTextAnchoredPosition>(HorizontalRelativeTo::Page,
12.f);
position.vertical =
boost::optional<VerticalTextAnchoredPosition>(VerticalRelativeTo::PageTopMargin,
 122.f);
frame.setPosition(position);
```

Используя типы смещения HorizontalRelativeTo.Column и VerticalRelativeTo.Page, можно установить абсолютное положение встроенного объекта в текстовом документе.

```
TextAnchoredPosition position = TextAnchoredPosition();
position.horizontal =
boost::optional<HorizontalTextAnchoredPosition>(HorizontalRelativeTo::Column,
125.f);
position.vertical =
boost::optional<VerticalTextAnchoredPosition>(VerticalRelativeTo::Page, 345.f);
frame.setPosition(position);
```

С помощью метода Frame::setDimensions() можно изменить размеры встроенных объектов

```
    Frame frame = inlineObject.getFrame();
    frame.setDimensions(Size<float>(50.0, 50.0));
```

Вариант обтекания текстом графического объекта `TextWrapType` может быть задан посредством использованием метода `Frame::setWrapType()`.

```
    Frame frame = inlineObject.getFrame();
    frame.setWrapType(TextWrapType::Inline);
```

4.1.5 Работа с таблицами текстового документа

В табличном документе таблицами являются страницы документа.

Доступ к объектам `Table` осуществляется из `Blocks` (см. Рисунок 7). В табличном документе таблицами являются листы документа.

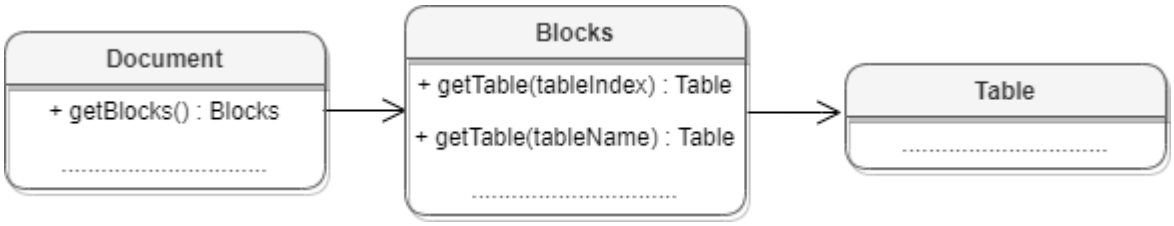


Рисунок 7 – Объектная модель для работы с таблицами

Получение таблицы текстового документа:

Для получения таблицы используется метод `Blocks::getTable()`. В качестве аргумента используется индекс или имя таблицы.

```
boost::optional<Table> table = document.getBlocks().getTable(0);
```

```
boost::optional<Table> table = document.getBlocks().getTable("Таблица1");
```

Перечисление таблиц текстового документа:

Для перечисления таблиц текстового документа можно использовать метод `Blocks::enumerateTables()`.

```
std::shared_ptr<Enumerator<Table>> tablesEnumerator =
document.getBlocks().getTablesEnumerator();
while (tablesEnumerator->isValid()) {
    Table table = tablesEnumerator->getCurrent();
    std::printf("%s", table.getName().c_str());
    tablesEnumerator->goToNext();
}
```

Вставка таблицы в текстовый документ:

Для вставки таблицы в текстовый документ используется метод `Position::insertTable()`. В качестве аргументов передаются размеры и имя таблицы.

```
Position position = document.getRange().getEnd();
position.insertTable(4, 3, "Таблица1");
```

Переименование таблицы:

Для переименования таблицы используется метод `Table::setName()`.

```
auto tableName = "Table1";
auto tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    table.setName(tableName);
    tableOpt = document.getBlocks().getTable(tableName);
}
```

Удаление таблицы:

Для удаления таблицы используется метод `Table::remove()`.

```
auto tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    table.remove();
}
```

4.1.6 Работа с закладками

Основным классом для работы с закладками является `Bookmarks`. Список закладок документа возвращает метод `Document::getBookmarks()`. Метод `Bookmarks::getBookmarkRange()` возвращает диапазон текста, метод `Bookmarks::removeBookmark()` удаляет закладку по имени. Для создания закладки используется метод `Position::insertBookmark()`.

Доступны следующие операции с закладками:

- вставка закладки в указанное местоположение;
- удаление закладки с заданным именем;
- поиск закладки по имени;
- замена текстового содержимого закладки;

- вставка текста в закладку;
- удаление содержимого закладки;
- получение текстового содержимого закладки;
- вставка таблицы в закладку.

Вставка закладки в указанное местоположение

```
Position startDocument = document.getRange().getBegin();
startDocument.insertBookmark("Bookmark")
```

Удаление закладки с заданным именем

```
document.getBookmarks().removeBookmark("Bookmark");
```

Поиск закладки по имени

```
Bookmarks bookmarks = document.getBookmarks();
boost::optional<Range> bookmarkRange = bookmarks.getBookmarkRange("Bookmark");
```

Замена текстового содержимого закладки

```
Bookmarks bookmarks = document.getBookmarks();
boost::optional<Range> bookmarkRange = bookmarks.getBookmarkRange("Bookmark");
if (bookmarkRange.has_value()) {
    bookmarkRange.get().replaceText("New bookmark text");
}
```

Вставка текста в закладку

```
boost::optional<Range> bookmarkRange = bookmarks.getBookmarkRange("Bookmark");
if (bookmarkRange.has_value()) {
    bookmarkRange.get().getBegin().insertText("New bookmark text");
}
```

Удаление содержимого закладки

```
boost::optional<Range> bookmarkRange = bookmarks.getBookmarkRange("Bookmark");
if (bookmarkRange.has_value()) {
    bookmarkRange.get().getBegin().removeBackward()
}
```

Получение текстового содержимого закладки

```
boost::optional<Range> bookmarkRange = bookmarks.getBookmarkRange("Bookmark");
if (bookmarkRange.has_value()) {
    std::printf("Bookmark range text : %s",
```

```
bookmarkRange.get().extractText().c_str());
}
```

Вставка таблицы в закладку

```
Bookmarks bookmarks = document.getBookmarks();
boost::optional<Range> bookmarkRange = bookmarks.getBookmarkRange("Bookmark");
if (bookmarkRange.has_value()) {
    bookmarkRange.get().getEnd().insertTable(3, 3, "signers_list");
}
```

4.1.7 Рецензирование документов

Средства рецензирования документа доступны в текстовом редакторе, они позволяют выполнять следующие действия:

- помечать изменения, вносимые пользователем в текстовый документ (TrackedChange);
- ассоциировать текстовый комментарий с фрагментом текстового документа (Comments).

Данные механизмы используются на стадии рецензирования или согласования документа с последующим внесением замечаний. Функции объектной модели для работы со средствами рецензирования позволяют получить детальную информацию о каждом изменении: автор изменения, дата внесения изменения, оригинальный текст, измененный текст.

Для включения или отключения режима рецензирования используется метод Document::setChangesTrackingEnabled(). Для проверки текущего статуса данного режима используется метод Document::isChangesTrackingEnabled().

Пример:

```
document.setChangesTrackingEnabled(true);
std::printf("IsChangesTrackingEnabled:", document.isChangesTrackingEnabled());
```

Инструменты рецензирования применяются к диапазону документа, по этой причине методы доступа к ним находятся в классе Range (см. Рисунок 8).

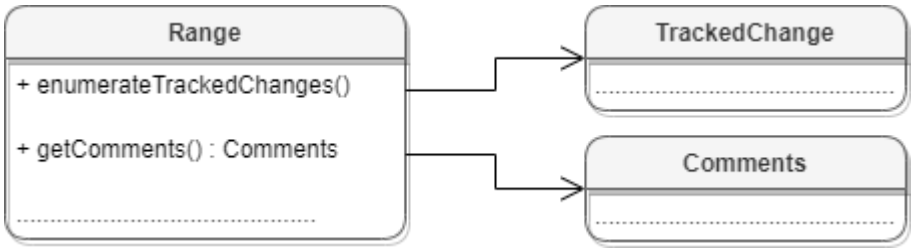


Рисунок 8 – Инструменты рецензирования документа

4.1.8 Поиск в текстовом документе

Для поиска в документе необходимо создать экземпляр класса `Search` посредством вызова `createSearch()`, затем использовать метод `Search::findText` (см. Рисунок 9).

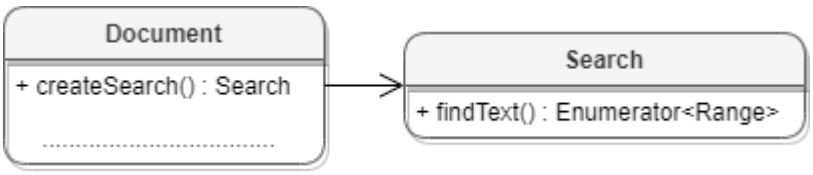


Рисунок 9 – Объектная модель для поиска в документе

Пример поиска в текстовом документе:

```
// Поиск в документе
auto search = CO::API::Document::createSearch(document);
auto searchResult = search->findText("Yellow");
while (searchResult->isValid())
{
    auto range = searchResult->getCurrent();
    std::printf(range.extractText().c_str());
    searchResult->goToNext();
}
```

Пример поиска в ячейке таблицы текстового документа:

```
// Поиск в ячейке E2 таблицы с индексом 0
Table firstTable = document.getBlocks().getTable(0).get();
Cell cell = firstTable.getCell("E2");

auto search = CO::API::Document::createSearch(document);
auto searchResult = search->findText("Yellow", cell.getRange());
while (searchResult->isValid())
{
    auto range = searchResult->getCurrent();
```

```
    std::printf(range.extractText().c_str());
    searchResult->goToNext();
}
```

4.2 Работа с табличным документом

4.2.1 Создание и открытие табличного документа

Метод Application::createDocument создает документ. В качестве параметра используется тип DocumentType. Для создания табличного документа необходимо выбрать тип DocumentType.Workbook.

Пример создания табличного документа:

```
auto document = application.createDocument(DocumentType::Text);
```

```
DocumentSettings documentSettings = DocumentSettings();
documentSettings.documentType = DocumentType::Workbook;
auto document = application.createDocument(documentSettings);
```

Метод Application::loadDocument открывает документ, находящийся по указанному пути.

Примеры загрузки табличного документа:

```
auto document = application.loadDocument("spreadsheet.docx");
```

```
auto documentSettings = DocumentSettings();
documentSettings.documentType = DocumentType::Text;
auto loadSettings = LoadDocumentSettings();
loadSettings.commonDocumentSettings = documentSettings;
auto document = application.loadDocument("spreadsheet.docx", loadSettings);
```

4.2.2 Сохранение и экспорт табличного документа

Метод Document::saveAs сохраняет документ по указанному пути.

Примеры сохранения табличного документа:

```
document.saveAs(filePath);
```

```
SaveDocumentSettings saveDocumentSettings = SaveDocumentSettings();


saveDocumentSettings.documentFormat = DocumentFormat::OXML;
saveDocumentSettings.documentType = DocumentType::Workbook;
```

```
saveDocumentSettings.documentPassword = "password";
saveDocumentSettings.isTemplate = false;

saveDocumentSettings.dsvSettings = DSVSettings();
saveDocumentSettings.dsvSettings.autofit = true;
saveDocumentSettings.dsvSettings.startBlockIndex = 0;
saveDocumentSettings.dsvSettings.lastBlockIndex = 10;

document.saveAs(filePath, saveDocumentSettings);
```

Метод `Document::exportAs` экспортирует документ в файл по указанному пути с заданным форматом типа `ExportFormat`.

В настоящее время поддерживается только операция экспорта документа в формат PDF.

Примеры экспорта табличного документа:

```
document.exportAs(filePath, ExportFormat::PDFA1);
```

```
WorkbookExportSettings workbookSettings = WorkbookExportSettings();
workbookSettings.sheetNames = SheetNames();
workbookSettings.sheetNames.push_back("Лист2");
workbookSettings.printingScope = PrintingScope(PrintingScope::Type::PrintArea);
workbookSettings.pageProperties = PageProperties(100, 200);
workbookSettings.scale = 90;
document.exportAs(filePath, ExportFormat::PDFA1, workbookSettings);
```

4.2.3 Диаграммы

Работа с диаграммами реализована только в табличных документах. На рисунке 10 изображена объектная модель классов, относящихся к работе с диаграммами.

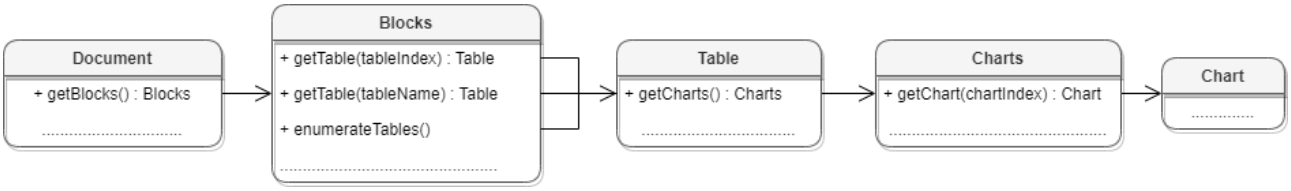


Рисунок 10 – Объектная модель классов для работы с диаграммами

Доступ к списку диаграмм производится через класс `Table`, соответствующий листу табличного документа.

Пример:

```
boost::optional<Table> sheetDocumentPage = document.getBlocks().getTable(0);
if (sheetDocumentPage.has_value()) {
    Charts charts = sheetDocumentPage.get().getCharts();
    std::printf("%d", charts.getChartsCount());
}
```

Для получения диаграммы `Chart` используется метод `Charts::getChart()`.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Charts charts = firstSheet.getCharts();
Chart chart = charts.getChart(0);
std::printf("Range:", chart.getRangeAsString());
```

Создание и удаление диаграмм в текущей версии не поддерживаются.

4.2.4 Поиск в табличном документе

Для поиска в документе необходимо создать экземпляр класса `Search` посредством вызова `createSearch()`, затем использовать метод `Search::findText` (см. Рисунок 11).

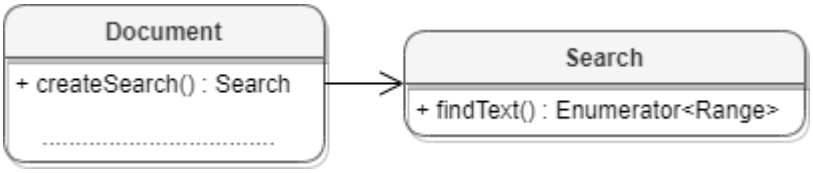


Рисунок 11 – Объектная модель для поиска в документе

Пример поиска в табличном документе:

```
// Поиск в документе
auto search = CO::API::Document::createSearch(document);
auto searchResult = search->findText("Yellow");
while (searchResult->isValid())
{
    auto range = searchResult->getCurrent();
    std::printf(range.extractText().c_str());
```

```
    searchResult->goToNext();
}
```

Пример поиска в ячейке табличного документа:

```
// Поиск в ячейке E2 страницы L1
Table firstSheet = document.getBlocks().getTable("L1").get();
Cell cell = firstSheet.getCell("E2");

auto search = CO::API::Document::createSearch(document);
auto searchResult = search->findText("Yellow", cell.getRange());
while (searchResult->isValid())
{
    auto range = searchResult->getCurrent();
    std::printf(range.extractText().c_str());
    searchResult->goToNext();
}
```

4.2.5 Работа с графическими объектами

На данный момент работа с изображениями в табличном документе не поддерживается.

4.2.6 Работа с листами табличного документа

В табличном документе таблицами являются страницы документа.

Доступ к объектам `Table` осуществляется из `Blocks` (см. Рисунок 12). В табличном документе таблицами являются листы документа.

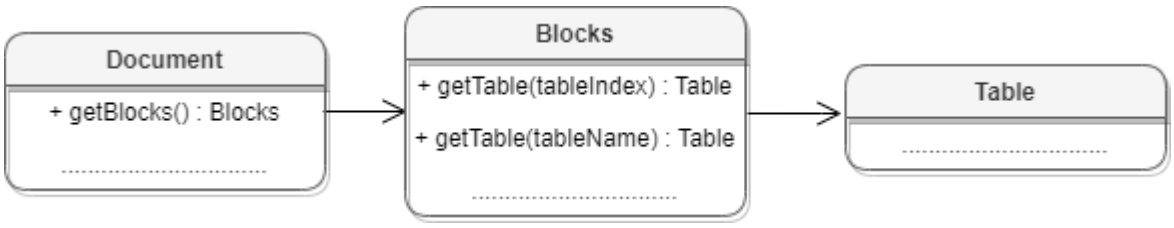


Рисунок 12 – Объектная модель для работы с таблицами

Получение листа табличного документа:

Для получения листа табличного документа используется метод `Blocks::getTable()`. В качестве аргумента используется индекс или имя таблицы.

```
boost::optional<Table> table = document.getBlocks().getTable(0);
```

```
boost::optional<Table> table = document.getBlocks().getTable("Таблица1");
```

Перечисление страниц табличного документа:

Для перечисления листов табличного документа можно использовать метод `Blocks::enumerateTables()`.

```
std::shared_ptr<Enumerator<Table>> tablesEnumerator =
document.getBlocks().getTablesEnumerator();
while (tablesEnumerator->isValid()) {
    Table table = tablesEnumerator->getCurrent();
    std::printf("%s", table.getName().c_str());
    tablesEnumerator->goToNext();
}
```

Также доступен вариант перечисления листов документа посредством использования метода `Blocks::getEnumerator()` с дальнейшим преобразованием блока в таблицу.

```
Blocks blocks = document.getBlocks();
std::shared_ptr<Enumerator<Block>> blocksEnumerator = blocks.getEnumerator();
if (blocksEnumerator) {
    while (blocksEnumerator->isValid()) {
        Block block = blocksEnumerator->getCurrent();
        boost::optional<Table> tableOpt = block.toTable();
        if (tableOpt.has_value()) {
            std::printf("%s", tableOpt.get().getName().c_str());
        }
        blocksEnumerator->goToNext();
    }
}
```

Вставка страницы в табличный документ:

Для вставки листа в табличный документ используется метод `Position::insertTable()`. В качестве аргументов передаются размеры и имя таблицы.

```
Position position = document.getRange().getEnd();
position.insertTable(4, 3, "Таблица1");
```

Переименование страницы:

Для переименования таблицы используется метод `Table::setName()`.

```
auto tableName = "Table1";
auto tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    table.setName(tableName);
    tableOpt = document.getBlocks().getTable(tableName);
}
```

Скрытие и отображение страниц табличного документа:

Для скрытия / отображения листа документа используется метод `Table::setVisible()`.

```
auto tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    table.setVisible(false);
}
```

Копирование страницы:

Для создания копии страницы используется метод `Table::duplicate()`.

```
auto tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    table.dulpicate();
}
```

Удаление страницы:

Для удаления таблицы используется метод `Table::remove()`.

```
auto tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    table.remove();
}
```

4.2.7 Работа со сводными таблицами

Сводная таблица - инструмент обработки данных, служащий для их обобщения и удобства обработки. Схема взаимодействия объектов, связанных со сводными таблицами, приведена на рисунке 13.

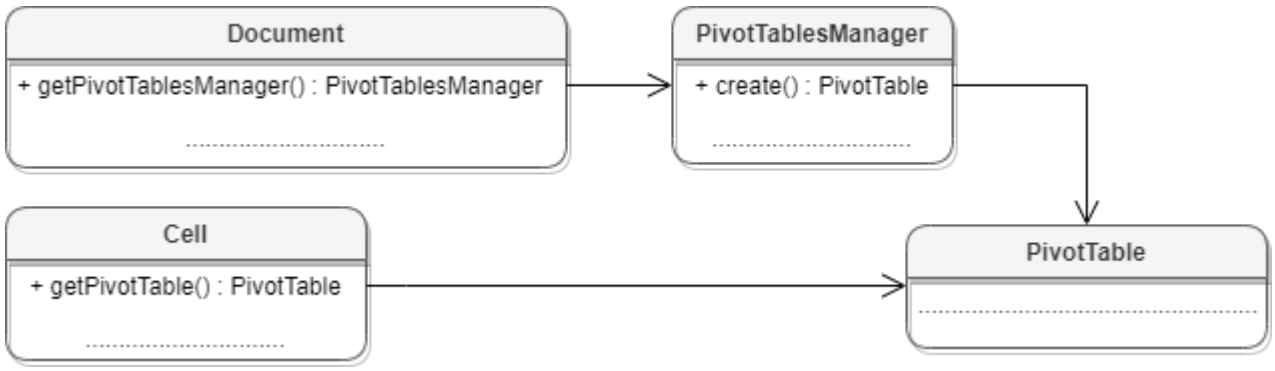


Рисунок 13 – Сводные таблицы

4.2.7.1 Получение сводной таблицы

Для получения диапазона исходных данных сводной таблицы используется метод `Cell::getPivotTable()`.

Пример:

```
auto tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    // Получаем ячейку, находящуюся в диапазоне исходных данных сводной таблицы
    auto table = tableOpt.get();
    auto pivotRootCell = table.getCell(CellPosition(2, 0));

    // Получаем сводную таблицу
    auto pivotTableOpt = pivotRootCell.getPivotTable();
    if (pivotTableOpt.has_value()) {
        ..............................
    }
}
```

4.2.7.2 Получение диапазона исходных данных сводной таблицы

Для получения диапазона исходных данных сводной таблицы используется метод `PivotTable::getSourceRange()`.

Пример:

```
// Получаем диапазон исходных данных сводной таблицы
auto sourceCellRange = pivotTable.getSourceRange();

// Для получения границ диапазона используем поля CellRange:
std::printf("Begin row", sourceCellRange.getBeginRow());
std::printf("Begin column", sourceCellRange.getBeginColumn());
std::printf("Last row", sourceCellRange.getLastRow());
std::printf("Last column", sourceCellRange.getLastColumn());
```

4.2.7.3 Получение диапазона размещения сводной таблицы

Для получения диапазона разещения сводной таблицы используется метод PivotTable::getPivotRange().

Пример:

```
// Получаем диапазон исходных данных сводной таблицы
auto pivotRange = pivotTable.getPivotRange();
std::printf("%d %d", pivotRange.getBeginColumn(), pivotRange.getLastColumn());
```

4.2.7.4 Получение неподдерживаемых свойств сводной таблицы

Для получения неподдерживаемых свойств сводной таблицы используется метод PivotTable::getUnsupportedFeatures().

Пример:

```
PivotTableUnsupportedFeatures pivotTableUnsupportedFeatures =
pivotTable.getUnsupportedFeatures();
for (int i = 0; i < pivotTableUnsupportedFeatures.size(); i++) {
    PivotTableUnsupportedFeature feature = pivotTableUnsupportedFeatures[i];
    std::printf("%d", feature);
}
```

4.2.7.5 Получение флагов отображения общих итогов для строк и колонок

Для получения флагов отображения общих итогов для строк и колонок используются методы PivotTable::isRowGrandTotalEnabled(), PivotTable::isColumnGrandTotalEnabled().

Пример:

```
// Получаем флаги отображения общих итогов для строк и колонок
std::printf("%d", pivotTable.isRowGrandTotalEnabled());
std::printf("%d", pivotTable.isColumnGrandTotalEnabled());
```

4.2.7.6 Получение заголовков сводной таблицы

Для получения заголовков сводной таблицы используется метод PivotTable::getPivotTableCaptions().

Пример:

```
PivotTableCaptions pivotTableCaptions = pivotTable.getPivotTableCaptions();
std::printf("%s", pivotTableCaptions.errorCaption.get().c_str());
std::printf("%s", pivotTableCaptions.emptyCaption.get().c_str());
std::printf("%s", pivotTableCaptions.grandTotalCaption.c_str());
std::printf("%s", pivotTableCaptions.valuesHeaderCaption.c_str());
std::printf("%s", pivotTableCaptions.columnHeaderCaption.c_str());
std::printf("%s", pivotTableCaptions.rowHeaderCaption.c_str());
```

4.2.7.7 Получение и применение фильтра для сводной таблицы

Для работы с фильтрами сводной таблицы используются методы PivotTable::getFilter(), PivotTableEditor::setFilter().

Пример:

```
// По названию поля сводной таблицы получаем фильтр
auto filterOpt = pivotTable.getFilter("Category");
if (filterOpt.has_value()) {
    auto filter = filterOpt.get();
    // Делаем элементы `Car` и `Technology` скрытыми
    filter.setHidden("Car", true);
    filter.setHidden("Technology", true);

    // Делаем элемент `Furniture` видимым
    filter.setHidden("Furniture", false);

    // Применяем фильтр к сводной таблице
    pivotTable.createPivotTableEditor().setFilter(filter).apply();
}
```

4.2.7.8 Получение полей из области фильтров

Для получения полей из области фильтров используется метод PivotTable::getPageFields().

Пример:

```
boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
PivotTablesManager pivotTablesManager = pivotTablesManagerOpt.get();
PivotTable pivotTable = pivotTablesManager.create(cellRange);
PivotTableCategoryFields pageFields = pivotTable.getPageFields();
for (int i = 0; i < pageFields.size(); i++) {
    std::printf("%s", pageFields[i].fieldProperties.fieldAlias.get());
    std::printf("%s", pageFields[i].fieldProperties.subtotalAlias.get());
    std::printf("%d", pageFields[i].fieldProperties.fieldName.c_str());
}
```

4.2.7.9 Получение полей из области значений

Для получения полей из области значений используется метод PivotTable::getValueFields().

Пример:

```
boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
PivotTablesManager pivotTablesManager = pivotTablesManagerOpt.get();
PivotTable pivotTable = pivotTablesManager.create(cellRange);
PivotTableValueFields valueFields = pivotTable.getValueFields();
for (int i = 0; i < valueFields.size(); i++) {
    std::printf("%s", valueFields[i].baseFieldName.c_str());
    std::printf("%d", valueFields[i].cellNumberFormat);
    std::printf("%s", valueFields[i].customFormula.get().c_str());
    std::printf("%d", valueFields[i].totalFunction);
    std::printf("%s", valueFields[i].valueFieldName.c_str());
}
```

4.2.7.10 Получение полей из области строк

Для получения полей из области строк используется метод PivotTable::getRowFields().

Пример:

```
PivotTableCategoryFields pivotTableRowFields = pivotTable.getRowFields();
for (int i = 0; i < pivotTableRowFields.size(); i++) {
    PivotTableCategoryField field = pivotTableRowFields[i];
    std::printf("%s", field.fieldProperties.fieldAlias.get().c_str());
    std::printf("%s", field.fieldProperties.subtotalAlias.get().c_str());
    std::printf("%s", field.fieldProperties.fieldName.c_str());
    PivotTableFunctions subtotalFunctions = field.subtotalFunctions;
    for (int j = 0; j < subtotalFunctions.size(); j++) {
        PivotTableFunction subtotalFunction = subtotalFunctions[j];
        std::printf("%d", subtotalFunction);
    }
}
```

4.2.7.11 Получение полей из области колонок

Для получения полей из области колонок используется метод `PivotTable::getColumnFields()`.

Пример:

```
boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
PivotTablesManager pivotTablesManager = pivotTablesManagerOpt.get();
PivotTable pivotTable = pivotTablesManager.create(cellRange);
PivotTableCategoryFields columnFields = pivotTable.getColumnFields();
for (int i = 0; i < columnFields.size(); i++) {
    std::printf("%s", columnFields[i].fieldProperties.fieldAlias.get());
    std::printf("%s", columnFields[i].fieldProperties.subtotalAlias.get());
    std::printf("%d", columnFields[i].fieldProperties.fieldName.c_str());
    for (int j = 0; j < columnFields[i].subtotalFunctions.size(); j++) {
        std::printf("%d", columnFields[i].subtotalFunctions[j]);
    }
}
```

4.2.7.12 Получение настроек отображения сводной таблицы

Для получения настроек отображения сводной таблицы используется метод `PivotTable::getPivotTableLayoutSettings()`.

Пример:

```
PivotTableLayoutSettings pivotTableLayoutSettings =
pivotTable.getPivotTableLayoutSettings();
std::printf("%d", pivotTableLayoutSettings.displayFieldCaptions);
std::printf("%d", pivotTableLayoutSettings.indentForCompactLayout);
std::printf("%d", pivotTableLayoutSettings.isMergeAndCenterLabelsEnabled);
std::printf("%d", pivotTableLayoutSettings.pageFieldOrder);
std::printf("%d", pivotTableLayoutSettings.pageFieldWrapCount);
std::printf("%d", pivotTableLayoutSettings.reportLayout);
std::printf("%d", pivotTableLayoutSettings.useGridDropZones);
std::printf("%d", pivotTableLayoutSettings.valueFieldsOrientation);
```

4.2.7.13 Обновление сводной таблицы

Для обновления сводной таблицы используется метод `PivotTable::update()`. Метод возвращает значение типа `PivotTableUpdateResult`.

```
// Пересчет и перезаполнение сводной таблицы в соответствии с исходными данными.
// Обновление сводной таблицы приводит к потере всех неподдерживаемых свойств.
PivotTableUpdateResult updateResult = pivotTable.update();
if (updateResult == PivotTableUpdateResult::FieldAlreadyEnabled) {
    ............
}
```

4.3 Работа с макрокомандами

Класс `Scripts` предоставляет доступ к списку макрокоманд документа. На рисунке 14 изображена объектная модель классов, относящихся к работе с макрокомандами.

Класс `Scripts` предназначен для доступа к списку макрокоманд, доступен через метод `Document::getScripts()`, класс `Scripting` служит для запуска макрокоманд, доступна через `Scripting::createScripting(document)`.

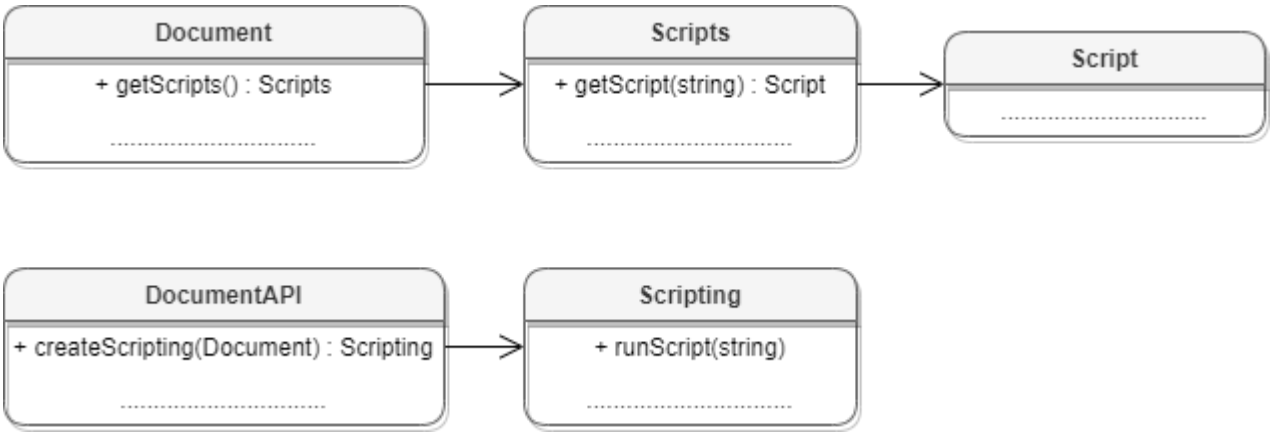


Рисунок 14 – Объектная модель классов для работы с макрокомандами

Доступны следующие операции:

- получение списка макрокоманд;
- добавление макрокоманды;
- получение макрокоманды по имени;
- удаление макрокоманды;
- запуск макрокоманды.

4.4 Работа с именованными диапазонами

Именованный диапазон – это диапазон ячеек или формула, которым присвоено имя. Преимуществом именованного диапазона является его информативность. Именованные диапазоны упрощают работу с ячейками, также их удобно использовать при работе с формулами. На данный момент доступна возможность работы с именованными диапазонами, представляющими собой ссылки на диапазоны ячеек. Доступ к именованным диапазонам осуществляется посредством методов `Document::getNamedExpressions()` и `Table::getNamedExpressions()` (см. Рисунок 15).

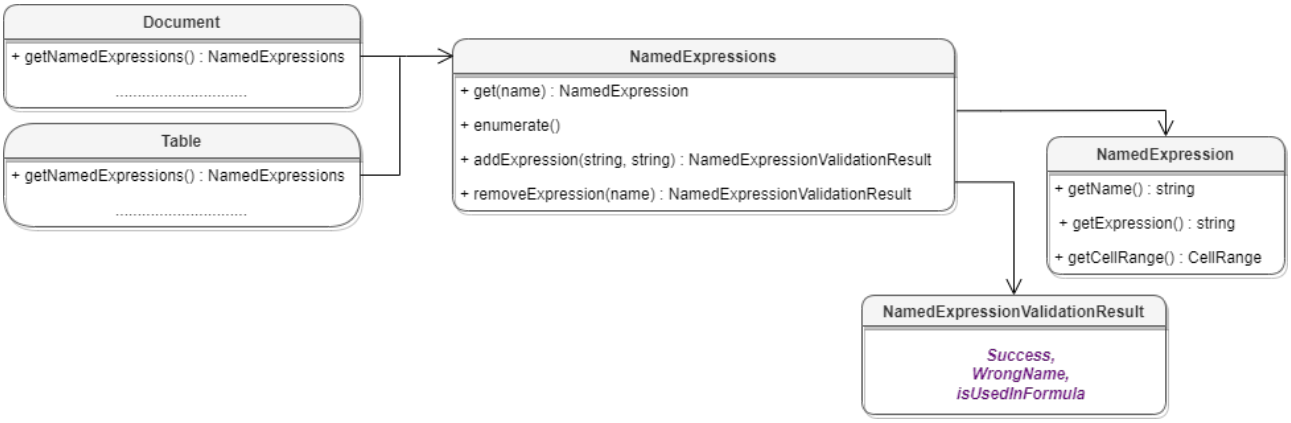


Рисунок 15 – Классы для работы с именованными диапазонами

4.4.1 Доступ к именованным диапазонам

Доступ к именованным диапазонам осуществляется посредством методов Document::getNamedExpressions() и Table::getNamedExpressions().

Примеры:

```
NamedExpressions namedExpressions = document.getNamedExpressions();
```

```
std::shared_ptr<Enumerator<Table>> blocksEnumerator =
document.getBlocks().getTablesEnumerator();
while (blocksEnumerator->isValid()) {
    Table table = blocksEnumerator->getCurrent();
    NamedExpressions namedExpressions = table.getNamedExpressions();
    std::shared_ptr<Enumerator<NamedExpression>> enumerator =
namedExpressions.getEnumerator();
    ..................................
    blocksEnumerator->goToNext();
}
```

4.4.2 Получение коллекции именованных диапазонов

Для перечисления именованных диапазонов используется объект NamedExpressionsEnumerator, который может быть получен с помощью метода Метод NamedExpressions::enumerate().

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
NamedExpressions namedExpressions = firstSheet.getNamedExpressions();
std::shared_ptr<Enumerator<NamedExpression>> enumerator =
namedExpressions.getEnumerator();
while (enumerator->isValid()) {
    NamedExpression namedExpression = enumerator->getCurrent();
    std::printf("%s", namedExpression.getName().c_str());
    enumerator->goToNext();
}
```

4.4.3 Получение свойств именованного диапазона

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
NamedExpressions namedExpressions = firstSheet.getNamedExpressions();
```

```
auto namedExpressionOpt = namedExpressions.get("Alice_Age");
if (namedExpressionOpt.has_value()) {
    auto namedExpression = namedExpressionOpt.get();
    auto name = namedExpression.getName();
    auto formula = namedExpression.getExpression();
    auto range = namedExpression.getCellRange();
}
```

4.4.4 Добавление именованного диапазона

Для добавления именованного диапазона используется метод NamedExpressions::addExpression(). В качестве результата операции метод возвращает значение типа NamedExpressionsValidationResult.

Пример:

```
NamedExpressions namedExpressions = firstSheet.getNamedExpressions();
std::string expressionName = "Покупки";
std::string expressionValue = "=Формула покупки!$E$6:$E$14";
NamedExpressionsValidationResult validationResult =
namedExpressions.addExpression(expressionName, expressionValue);
std::printf("%d", validationResult);
```

4.4.5 Удаление именованного диапазона

Для удаления именованного диапазона используется метод NamedExpressions::removeExpression(). В качестве результата операции метод возвращает значение типа NamedExpressionsValidationResult.

Пример:

```
std::string expressionName = "Покупки";
boost::optional<NamedExpression> namedExpressionOpt =
namedExpressions.get(expressionName);
if (namedExpressionOpt.has_value()) {
    NamedExpressionsValidationResult validationResult =
namedExpressions.removeExpression(expressionName);
    std::printf("%d", validationResult);
}
```

4.4.6 Получение параметров именованного диапазона

Для получения детальной информации об именованном диапазоне используются методы NamedExpression:getName, NamedExpression:getExpression,

NamedExpression:getCellRange.

```
auto name = namedExpression.getName();
auto formula = namedExpression.getExpression();
auto range = namedExpression.getCellRange();
```

4.5 Работа со строками и столбцами таблиц

4.5.1 Группировка строк и колонок таблицы

Следующий набор методов позволяет группировать строки и колонки таблицы: Table::groupRows(), Table::ungroupRows(), Table::clearRowGroups(), Table::groupColumns(), Table::ungroupColumns(), Table::clearColumnnGroups().

Редактор дает возможность отображать группы в виде иерархии. Совместно с данными методами можно использовать методы Table::setColumnsVisible и Table::setRowsVisible чтобы раскрывать и закрывать фрагменты иерархии групп.

Методы могут вызвать исключения DocumentAPI::OutOfRangeError и DocumentAPI::IncorrectArgumentError в случае использования индексов, выходящих за рамки таблицы.

4.5.2 Управление видимостью строк / колонок

Метод Table::setColumnsVisible позволяет задавать видимость столбцов, начиная с заданного индекса.

Метод Table:setRowsVisible позволяет задавать видимость строк, начиная с заданного индекса.

4.6 Работа с ячейками таблиц

4.6.1 Доступ к ячейкам

Доступ к ячейкам таблицы возможен двумя способами (см. Рисунок 16):

– непосредственно из таблицы, используя метод Table::getCell();

– из диапазона ячеек методом перечисления CellRange::enumerate().

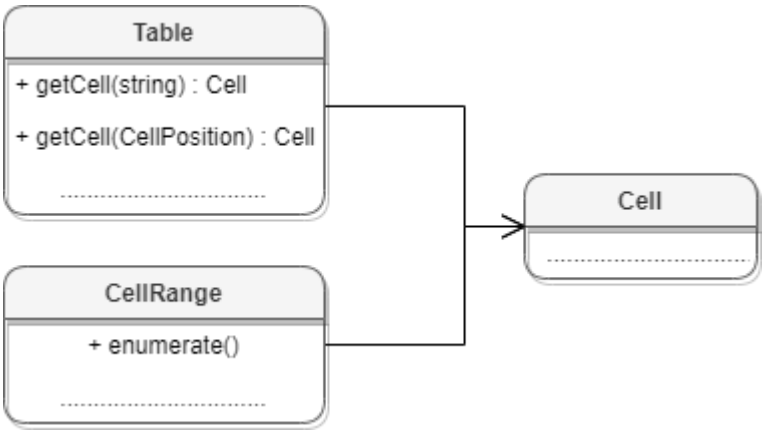


Рисунок 16 – Объектная модель для работы с ячейками таблиц

Для получения содержимого ячейки, заполнения данных, а также для форматирования ячейки используется объект `Cell`, представляющий ячейку таблицы с указанным адресом. Метод `Table::getCell()` возвращает экземпляр класса `Cell`.

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("B1");
```

Второй вариант доступа к ячейке - перечисление диапазона ячеек методом `CellRange::enumerate()`.

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
std::shared_ptr<Enumerator<Cell>> enumerator = cellRange.getEnumerator();
while (enumerator->isValid()) {
    Cell cell = enumerator->getCurrent();
    std::printf("%s", cell.getFormattedValue().c_str());
    enumerator->goToNext();
}
```

Для установки значений ячеек используются методы `Cell::setText`, `Cell::setNumber`, `Cell::setFormula`, `Cell::setBool`.

Примеры:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("B1");
cell.setText("Текст");
std::printf("%s", cell.getFormattedValue().c_str());
```

```
cell.setNumber(10);
std::printf("%s", cell.getFormattedValue().c_str());

cell.setFormula("=SUM(B2:B3)");
std::printf("%s", cell.getFormattedValue().c_str());

cell.setBool(false);
std::printf("%s", cell.getFormattedValue().c_str());

cell.setFormattedValue("12:39");
std::printf("%s", cell.getFormattedValue().c_str());
```

Для установки даты и времени используется метод `Cell::setFormattedValue`. Данная функция пытается определить тип значения, переданного в качестве аргумента (число, дата и т.д.) и применяет необходимое форматирование.

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("B1");
cell.setFormattedValue("22.07.2020");
std::printf("%s", cell.getFormattedValue().c_str());

cell.setFormattedValue("12:39");
std::printf("%s", cell.getFormattedValue().c_str());
```

При необходимости есть возможность явно указать формат вводимого значения `CellFormat` (процентный, денежный, экспоненциальный и т.д.), для этого используется функция `Cell::SetFormat()`.

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("B1");
cell.setFormat(CellFormat::Accounting);
cell.setNumber(12);
std::printf("%s", cell.getFormattedValue().c_str());
```

Для получения значения ячейки используется метод `Cell::getFormattedValue()`.

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("B1");
std::printf("%s", cell.getFormattedValue().c_str());
```

4.6.2 Форматирование ячеек

При работе с ячейками таблиц можно использовать следующие варианты форматирования:

– форматирование параметров ячейки `CellProperties`, например, цвет фона, угол поворота текста;
– форматирование абзаца ячейки, например, отступы абзаца, межстрочный интервал текста;
– форматирование текста, например, цвет текста, начертание;
– задание параметров границ ячеек.

Содержимое ячейки (контент), вне зависимости от того является ли оно текстом, числовым значением или формулой, также описывается экземпляром класса Paragraph, и обладает свойствами `ParagraphProperties`. Это дает возможность управлять настройками отображения контента как отдельного абзаца, так и группы абзацев (например, если ячейка содержит несколько предложений текста). Для управления этим настройками используются методы `Cell::getParagraphProperties()` и `Cell::setParagraphProperties()`.

Пример установки и получения свойств параграфа ячейки:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("A2");

ParagraphProperties paragraphProperties = cell.getParagraphProperties();
paragraphProperties.alignment = Alignment::Center;
cell.setParagraphProperties(paragraphProperties);
```

Управление настройками текста ячейки (шрифт, цвет) производится через соответствующий ему диапазон. Класс Cell позволяет получить диапазон для всего контента с помощью метода `Cell::getRange()`. Далее, метод `Range::getTextProperties()` позволяет получить экземпляр класса `TextProperties`, представляющий свойства текста. После изменения значения свойств их необходимо применить к тексту ячейки с помощью метода `Range::setTextProperties()`.

Пример настроек текста ячейки:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell(CellPosition(0,1));

TextProperties textProperties = cell.getRange().getTextProperties();
textProperties.bold = true;
textProperties.italic = true;
textProperties.textColor = Color(ColorRGBA(55, 146, 179, 200));

cell.getRange().setTextProperties(textProperties);
```

4.6.3 Форматирование границ ячеек

Для оформления границ ячеек используется класс `Borders` (см. Рисунок 17). Он описывает свойства полей, соответствующих границам и диагоналям ячейки: `Left`, `Right`, `Top`, `Bottom`, `DiagonalDown`, `DiagonalUp`, `InnerHorizontal`, `InnerVertical`. Каждая граница ячейки описывается классом `LineProperties`, который, в свою очередь, обладает свойствами `LineStyle`, `LineEndingProperties`, `Color`, `LineWidth`.

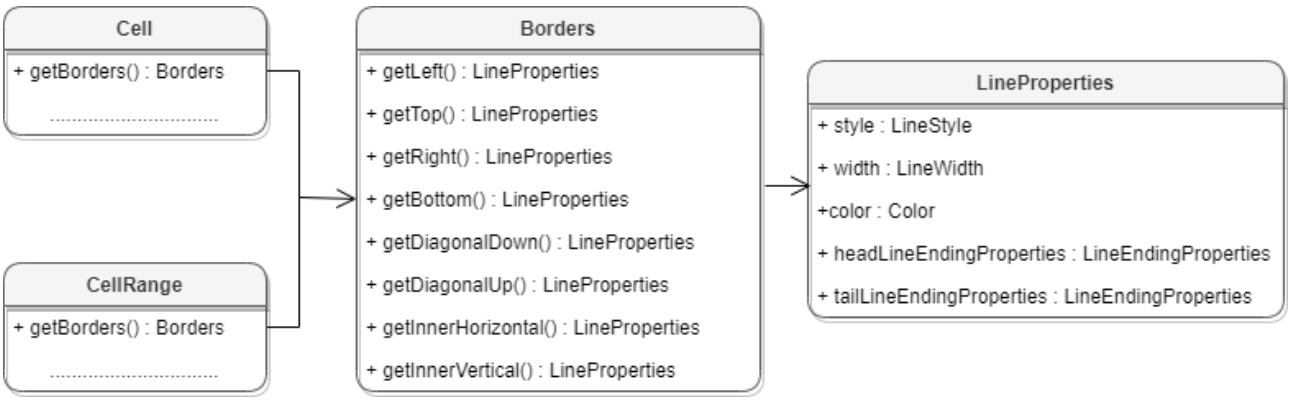


Рисунок 17 – Классы для работы с границами ячеек

Для оформления границ отдельной ячейки или группы ячеек необходимо выполнить следующие действия:

– получить ячейку `Cell` или область ячеек `CellRange`;
– настроить параметры для рисования линии границы с помощью экземпляра класса `LineProperties`;
– настроить свойства линии: левой границы, верхней границы и т.д. с помощью экземпляра класса `Borders`;

– установить границы ячеек с помощью Cell::setBorders() или CellRange::setBorders().

Пример настройки границ ячеек:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
CellRange cellRange = firstSheet.getCellRange("F3:H7");

LineProperties lineProperties = LineProperties();
lineProperties.style = LineStyle::Solid;
lineProperties.width = 1.5;
lineProperties.color = Color(ColorRGBA(55, 146, 179, 200));

Borders borders = Borders();
borders.setOuter(lineProperties);

cellRange.setBorders(borders);
```

4.6.4 Объединение и разделение ячеек таблицы

Допустимо объединение произвольного числа ячеек таблицы. При объединении указанный диапазон становится единой ячейкой. После завершения операции объединенная ячейка получает значение первой ячейки диапазона.

Для объединения нескольких ячеек используйте метод CellRange::merge().

Пример:

```
// Объединение ячеек A1 и A2 на первом листе табличного документа
Table firstSheet = document.getBlocks().getTable(0).get();
firstSheet.getCellRange("A1:A2").merge();
```

Допустимо разъединение только тех ячеек, которые были объединены ранее. После завершения операции данные, содержавшиеся в объединенной ячейке, будут помещены в верхнюю левую ячейку диапазона.

Для разъединения ячеек используйте метод CellRange::unmerge().

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
// Ячейка A1 является результатом объединения диапазона A1:A2
firstSheet.getCell("A1").unmerge()
```

5 Глобальные методы

5.1 Метод createSearch

Метод инициализирует механизм поиска для текущего документа. Возвращает объект `Search`, с помощью которого выполняются поисковые запросы.

Пример:

```
std::shared_ptr<Search> search = CO::API::Document::createSearch(document);
std::shared_ptr<Enumerator<Range>> searchResult = search->findText("API");
```

6 Справочник классов, структур и методов

Далее приведено описание классов, структур и методов библиотеки MyOffice Document API для языка программирования C++.

6.1 Класс AccountingCellFormatting

Класс содержит параметры финансового формата ячеек таблицы и используется в качестве аргумента метода `Cell::setFormat()`.

Описание полей класса `AccountingCellFormatting` представлено в таблице 2.

Таблица 2 – Описание полей класса `AccountingCellFormatting`

Поле	Описание
`AccountingCellFormatting.decimalPlaces`	Количество десятичных позиций
`AccountingCellFormatting.symbol`	Символ денежной единицы
`AccountingCellFormatting.localeCode`	Идентификатор кода языка (MS-LCID)
`AccountingCellFormatting.fillSymbol`	Символ заполнения
`AccountingCellFormatting.useThousandsSeparator`	Использовать разделитель для тысячных
`AccountingCellFormatting.currencySignPlacement`	Тип размещения знака валюты `CurrencySignPlacement`

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("A2");

AccountingCellFormatting cellFormat = AccountingCellFormatting();
cellFormat.decimalPlaces = 2;
cellFormat.symbol = "Руб";

cell.setFormat(cellFormat);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.2 Класс Alignment

Тип `Alignment` содержит варианты горизонтального выравнивания текста, в том числе в ячейке таблицы.

```
enum class Alignment
{
    Default,
    Left,
    Center,
    Right,
    Justify,
    Distributed,
    Fill
};
```

Варианты горизонтального выравнивания текста:

- `Default` – выравнивание текста по умолчанию;
- `Left` – выравнивание текста по левому краю;
- `Center` – выравнивание текста по центру;
- `Right` – выравнивание по правому краю;
- `Justify` – выравнивание по ширине;
- `Distributed` – распределенное выравнивание, при применении которого между словами добавляются пробелы так, чтобы оба края каждой строки были выровнены по обеим сторонам. Последняя строка в абзаце также выравнивается по обеим сторонам, но если строка состоит из одного слова, то выравнивание по правой стороне не осуществляется;
- `Fill` – распределение текста по горизонтали – заполнение строки текстом.

Пример:

```
ParagraphProperties paragraphProperties = paragraph.getParagraphProperties();
paragraphProperties.alignment = Alignment::Center;
```

6.3 Класс AnchoredPosition

Класс `AnchoredPosition` является оберткой (wrapper) над `TextAnchoredPosition`, содержит конструктор, принимающий в качестве аргумента объект `TextAnchoredPosition`, а также поле `textPosition` типа `boost::optional<TextAnchoredPosition>`.

Пример:

```
AnchoredPosition anchoredPosition = AnchoredPosition(textAnchoredPosition);
boost::optional<TextAnchoredPosition> textAnchoredPositionOpt =
anchoredPosition.textPosition;
if (textAnchoredPositionOpt.has_value()) {
    textAnchoredPosition = textAnchoredPositionOpt.get();
}
```

6.4 Класс Application

Класс `Application` управляет параметрами и объектами приложения. Предоставляет интерфейс для создания и загрузки документов. Допустимо использование только одного объекта `Application` для всего сеанса обработки документа.

6.4.1 Метод Application::createDocument

Метод `Application::createDocument` создает новый документ с типом `DocumentType`, либо `DocumentSettings`, возвращает объект `Document`.

Используется один из следующих вариантов метода:

```
application.createDocument(documentType: DocumentType) : Document
application.createDocument(documentSettings: DocumentSettings) : Document
```

Примеры:

```
auto document = application.createDocument(DocumentType::Text);
document.saveAs("NewTextDocument.xodt");
```

```
DocumentSettings documentSettings = DocumentSettings();
documentSettings.documentType = DocumentType::Workbook;
auto document = application.createDocument(documentSettings);
document.saveAs("NewSheetDocument.xlsx");
```

6.4.2 Метод Application::loadDocument

Метод `Application::loadDocument` загружает существующий текстовый или табличный документ из файла, находящегося по указанному пути. Формат и тип документа определяются из расширения файла, если не указаны явно с помощью параметра `LoadDocumentSettings`. Метод возвращает объект `Document`.

Используется один из следующих вариантов метода:

```
application.loadDocument(path: String) : Document
application.loadDocument(path: String, loadSettings: LoadDocumentSettings) : Document
```

Примеры загрузки текстового документа:

```
auto document = application.loadDocument("spreadsheet.xlsx");
```

```
auto documentSettings = DocumentSettings();
documentSettings.documentType = DocumentType::Workbook;
auto loadSettings = LoadDocumentSettings();
loadSettings.commonDocumentSettings = documentSettings;
auto document = application.loadDocument("spreadsheet.xlsx", loadSettings);
```

Примеры загрузки табличного документа:

```
auto document = application.loadDocument("spreadsheet.docx");
```

```
auto documentSettings = DocumentSettings();
documentSettings.documentType = DocumentType::Text;
auto loadSettings = LoadDocumentSettings();
loadSettings.commonDocumentSettings = documentSettings;
auto document = application.loadDocument("spreadsheet.docx", loadSettings);
```

6.4.3 Метод Application::getMessenger

Метод `Application::getMessenger` возвращает объект `Messenger`, реализующий логирование событий.

Пример:

```
Messenger::MessageHandlerFunction handler;
std::shared_ptr<Messenger> messenger = application.getMessenger();
std::shared_ptr<Connection> connection = messenger->subscribe(handler);
```

6.5 Класс Block

Класс `Block` является базовым для всех блоков документа. От него наследуются классы `Paragraph`, `Table`, `Shape`, `Field` (см. Рисунок 18).

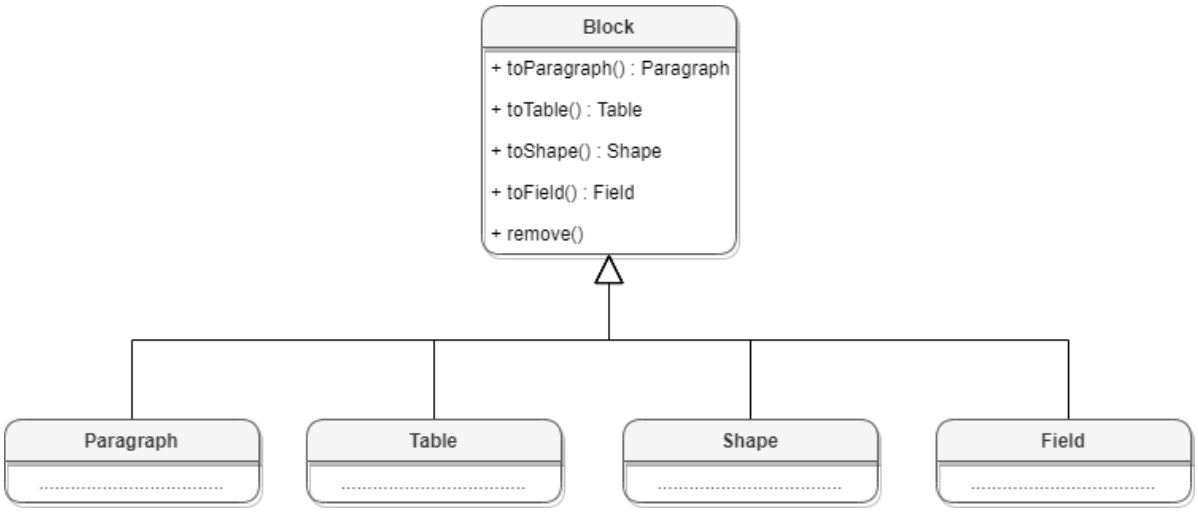


Рисунок 18 – Объектная модель класса `Block`

6.5.1 Методы toParagraph, toTable, toShape, toField

Преобразует объект `Block` в объект соответствующего типа.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Block> blockOpt = blocks.getBlock(0);
if (blockOpt.has_value()) {
    boost::optional<Paragraph> paragraph = blockOpt.get().toParagraph();
    if (paragraph.has_value()) {
        std::printf("%s", paragraph.get().getRange().extractText().c_str());
    }
}
```

6.5.2 Метод Block::getRange

Возвращает диапазон `Range`, в котором содержится данный блок.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Block> blockOpt = blocks.getBlock(0);
if (blockOpt.has_value()) {
    std::printf("%s", blockOpt.get().getRange().extractText().c_str());
}
```

6.5.3 Метод Block::remove

Удаляет блок из документа. Текущий экземпляр объекта `Block` становится недействительным.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Block> blockOpt = blocks.getBlock(0);
if (blockOpt.has_value()) {
    blockOpt.get().remove()
}
```

6.5.4 Метод Block::getSection

Метод возвращает раздел `Section`, содержащий блок.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Block> blockOpt = blocks.getBlock(0);
if (blockOpt.has_value()) {
    Section section = blockOpt.get().getSection();
    std::printf("%s", section.getRange().extractText().c_str());
}
```

6.6 Класс Blocks

Класс `Blocks` обеспечивает доступ к блокам `Block` документа или диапазона документа (см. Рисунок 19). Объект класса `Blocks` может быть получен вызовом метода `Document::getBlocks` или `HeaderFooter::getBlocks`.

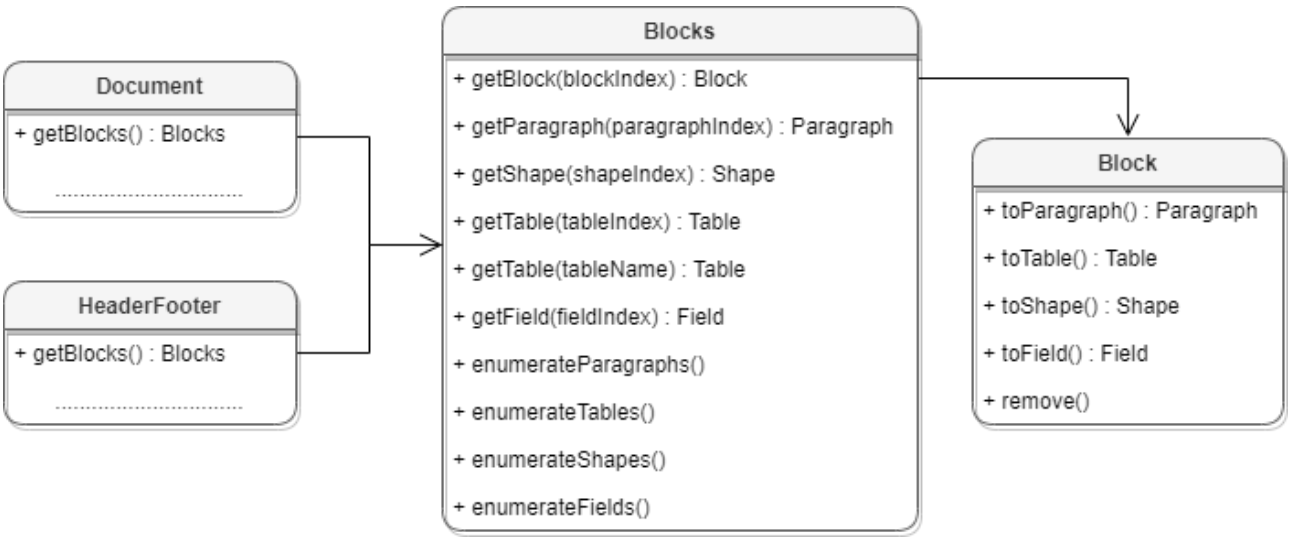


Рисунок 19 – Объектная модель класса `Blocks`

6.6.1 Метод Blocks::getBlock

Возвращает объект типа `Block` по заданному индексу. Нумерация индексов начинается с нуля.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Block> blockOpt = blocks.getBlock(0);
std::printf("%s", blockOpt.has_value() ? "Block was found" : "Block was not found");
```

6.6.2 Метод Blocks::getParagraph

Возвращает абзац с указанным индексом. Нумерация индексов начинается с нуля.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
std::printf("%s", paragraphOpt.has_value() ? "Paragraph was found" : "Paragraph was not found");
```

6.6.3 Метод Blocks::getTable

Для табличного документа возвращает лист (`worksheet`), для текстового документа возвращает таблицу. Параметры поиска - индекс или имя таблицы. Нумерация листов начинается с нуля.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Table> tableOpt = blocks.getTable(0);
std::printf("%s", tableOpt.has_value() ? "Table was found" : "Table was not found");
```

В качестве параметра метода также можно указать имя таблицы.

Пример:

```
boost::optional<Table> tableOpt = blocks.getTable("Sheet1");
```

6.6.4 Метод Blocks::getShape

Возвращает фигуру Shape по заданному индексу.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Shape> shapeOpt = blocks.getShape(0);
std::printf("%s", shapeOpt.has_value() ? "Shape was found" : "Shape was not
found");
```

6.6.5 Метод Blocks::getField

Возвращает объект типа Field по заданному индексу.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Field> fieldOpt = blocks.getField(0);
std::printf("%s", fieldOpt.has_value() ? "Field was found" : "Field was not
found");
```

6.6.6 Метод Blocks::getEnumerator

Позволяет перечислить объекты типа Block.

Пример:

```
Blocks blocks = document.getBlocks();
std::shared_ptr<Enumerator<Block>> blocksEnumerator = blocks.getEnumerator();
if (blocksEnumerator) {
    while (blocksEnumerator->isValid()) {
        Block block = blocksEnumerator->getCurrent();
        std::printf("%s", block.getRange().extractText().c_str());
        blocksEnumerator->goToNext();
    }
}
```

6.6.7 Метод Blocks::getParagraphsEnumerator

Позволяет реализовать перечисление абзацев Paragraph.

Пример:

```
Blocks blocks = document.getBlocks();
std::shared_ptr<Enumerator<Paragraph>> paragraphsEnumerator =
blocks.getParagraphsEnumerator();
if (paragraphsEnumerator) {
```

```
    while (paragraphsEnumerator->isValid()) {
        Paragraph paragraph = paragraphsEnumerator->getCurrent();
        std::printf("%s", paragraph.getRange().extractText().c_str());
        paragraphsEnumerator->goToNext();
    }
}
```

6.6.8 Метод Blocks::getTablesEnumerator

Позволяет перечислить объекты типа Table.

Пример:

```
Blocks blocks = document.getBlocks();
std::shared_ptr<Enumerator<Table>> tablesEnumerator =
blocks.getTablesEnumerator();
if (tablesEnumerator) {
    while (tablesEnumerator->isValid()) {
        Table table = tablesEnumerator->getCurrent();
        std::printf("%s", table.getRange().extractText().c_str());
        tablesEnumerator->goToNext();
    }
}
```

6.6.9 Метод Blocks::getShapesEnumerator

Позволяет перечислить объекты типа Shape.

Пример:

```
Blocks blocks = document.getBlocks();
std::shared_ptr<Enumerator<Shape>> shapesEnumerator =
blocks.getShapesEnumerator();
if (shapesEnumerator) {
    while (shapesEnumerator->isValid()) {
        Shape shape = shapesEnumerator->getCurrent();
        std::printf("%s", shape.getRange().extractText().c_str());
        shapesEnumerator->goToNext();
    }
}
```

6.6.10 Метод Blocks::getFieldsEnumerator

Позволяет перечислить объекты типа `Field`.

Пример:

```
Blocks blocks = document.getBlocks();
std::shared_ptr<Enumerator<Field>> fieldsEnumerator =
blocks.getFieldsEnumerator();
if (fieldsEnumerator) {
    while (fieldsEnumerator->isValid()) {
        Field field = fieldsEnumerator->getCurrent();
        std::printf("%s", field.getRange().extractText().c_str());
        fieldsEnumerator->goToNext();
     }
}
```

6.7 Класс Bookmarks

Предоставляет доступ к операциям с закладками в документе.

6.7.1 Метод Bookmarks::getBookmarkRange

Возвращает экземпляр объекта `Range` для дальнейшей работы с содержимым закладки.

Пример:

```
Bookmarks bookmarks = document.getBookmarks();
boost::optional<Range> bookmarkRange = bookmarks.getBookmarkRange("Bookmark");
if (bookmarkRange.has_value()) {
    bookmarkRange.get().replaceText("New bookmark text");
    std::printf("Bookmark range text : %s",
bookmarkRange.get().extractText().c_str());
}
```

6.7.2 Метод Bookmarks::removeBookmark

Удаляет закладку по ее названию.

Пример:

```
document.getBookmarks().removeBookmark("Bookmark");
```

6.8 Класс Borders

Класс `Borders` предназначен для оформления границ отдельной ячейки таблицы (см. таблицу 3). Параметры линии, такие как тип линии, ее ширина и цвет, задаются с помощью объектов типа `LineProperties`.

Таблица 3 – Описание методов класса `Borders`

Метод	Описание
`Borders:setLeft`	Установка левой границы ячейки
`Borders:setRight`	Установка правой границы ячейки
`Borders:setTop`	Установка верхней границы ячейки
`Borders:setBottom`	Установка нижней границы ячейки
`Borders:setDiagonalDown`	Установка диагональной линии 
`Borders:setDiagonalUp`	Установка диагональной линии 
`Borders:setOuter`	Установка внешних границ ячейки
`Borders:setDiagonals`	Установка обеих типов диагональных линий одновременно
`Borders:setInnerHorizontal`	Установка внутренних горизонтальных границ ячейки
`Borders:setInnerVertical`	Установка внутренних вертикальных границ ячейки
`Borders:setInner`	Установка внутренних границ ячейки
`Borders:setAll`	Установка всех границ ячейки
`Borders:getLeft`	Получение левой границы ячейки
`Borders:getRight`	Получение правой границы ячейки
`Borders:getTop`	Получение верхней границы ячейки
`Borders:getBottom`	Получение нижней границы ячейки
`Borders:getDiagonalDown`	Получение диагональной линии
`Borders:getDiagonalUp`	Получение диагональной линии
`Borders:getInnerHorizontal`	Получение внутренних горизонтальных границ ячейки
`Borders:getInnerVertical`	Получение внутренних вертикальных границ ячейки

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("C3");

LineProperties lineProperties = LineProperties();
```

```
lineProperties.style = LineStyle::Solid;
lineProperties.width = 1.5;
lineProperties.color = Color(ColorRGBA(55, 146, 179, 200));

Borders borders = Borders();
borders.setLeft(lineProperties);
borders.setTop(lineProperties);
borders.setRight(lineProperties);
borders.setBottom(lineProperties);
cell.setBorders(borders);
```

6.9 Класс Cell

Класс `Cell` предоставляет доступ к ячейке в таблице текстового документа или на листе табличного документа (см. Рисунок 20).

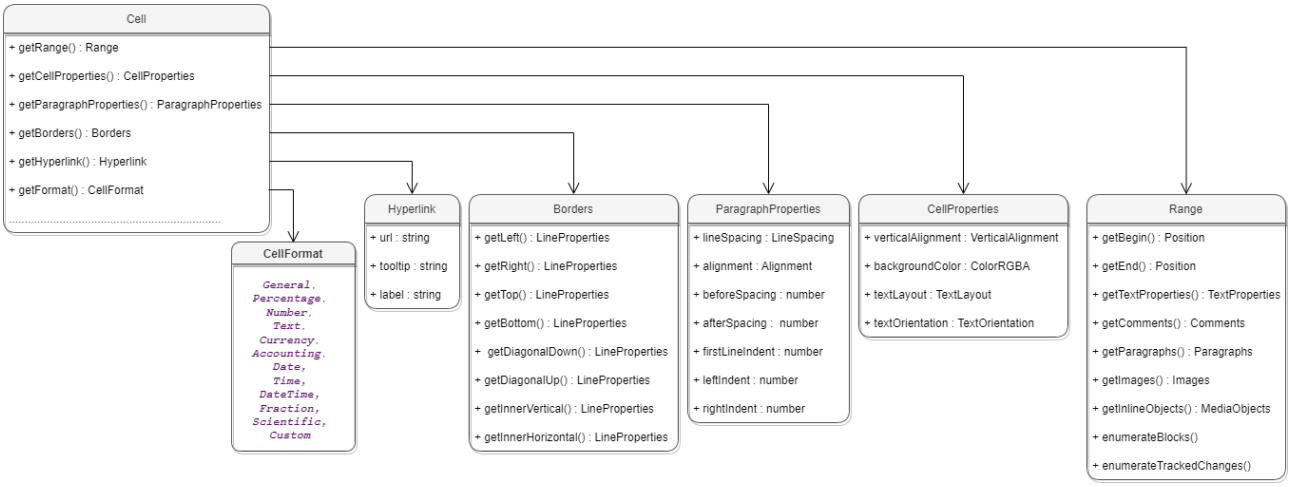


Рисунок 20 – Объектная модель ячейки таблиц

6.9.1 Метод Cell::getRange

Метод возвращает объект `Range` для управления содержимым ячейки.

6.9.2 Метод Cell::setBorders

Метод предназначен для установки границ ячейки. Примеры использования приведены в разделе `Borders`.

6.9.3 Метод Cell::setFormula

Метод позволяет вставить формулу в ячейку табличного документа.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
firstSheet.getCell("A3").setFormula("=SUM(A1:A2)");
```

6.9.4 Метод Cell::setFormat

Метод устанавливает формат ячейки. Существуют несколько вариантов использования метода.

Варианты вызова метода:

```
setFormat(cellFormat)
```

Где **cellFormat** – формат ячейки типа `CellFormat`.

```
setFormat(accountingCellFormatting)
```

Где **accountingCellFormatting** – формат ячейки типа `AccountingCellFormatting`.

```
setFormat(percentageCellFormatting)
```

Где **percentageCellFormatting** – формат ячейки типа `PercentageCellFormatting`.

```
setFormat(numberCellFormatting)
```

Где **numberCellFormatting** – формат ячейки типа `NumberCellFormatting`.

```
setFormat(currencyCellFormatting)
```

Где **currencyCellFormatting** – формат ячейки типа `CurrencyCellFormatting`.

```
setFormat(dateTimeCellFormatting, typeFormat)
```

Где **dateTimeCellFormatting** – формат ячейки типа `DateTimeCellFormatting`, **typeFormat** - формат даты/времени типа `CellFormat`.

```
setFormat(fractionCellFormatting)
```

Где **fractionCellFormatting** – формат ячейки типа `FractionCellFormatting`.

```
setFormat(scientificCellFormatting)
```

Где **scientificCellFormatting** – формат ячейки типа `ScientificCellFormatting`.

Примеры использования:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
cell.setNumber(2.3);

// Формат: Общий
cell.setFormat(CellFormat::General);
std::printf("General format: %d", cell.getFormat()); // 0
std::printf("Value: %d", cell.getRange().extractText().c_str()); // 2,3

// Формат : Процентный
PercentageCellFormatting percentageCellFormatting = PercentageCellFormatting();
percentageCellFormatting.decimalPlaces = 1;
cell.setFormat(percentageCellFormatting);
std::printf("Percentage format: %d", cell.getFormat()); // 1
std::printf("Value: %d", cell.getRange().extractText().c_str()); // 230,0%

// Формат : Числовой
NumberCellFormatting numberCellFormatting = NumberCellFormatting();
numberCellFormatting.decimalPlaces = 2;
cell.setFormat(numberCellFormatting);
std::printf("Numeric format: %d", cell.getFormat()); // 2
std::printf("Value: %d", cell.getRange().extractText().c_str()); // 2,30

// Формат : Денежный
CurrencyCellFormatting currencyCellFormatting = CurrencyCellFormatting();
currencyCellFormatting.symbol = "$";
cell.setFormat(currencyCellFormatting);
std::printf("Currency format: %d", cell.getFormat()); // 4
std::printf("Value: %d", cell.getRange().extractText().c_str()); // 2,30$

// Формат : Финансовый
AccountingCellFormatting accountingCellFormatting = AccountingCellFormatting();
accountingCellFormatting.symbol = "₽";
cell.setFormat(accountingCellFormatting);
std::printf("Accounting format: %d", cell.getFormat()); // 5
std::printf("Value: %d", cell.getRange().extractText().c_str()); // 2,30₽
```

```
// Формат : Дата / Время
DateTimeCellFormatting dateTimeCellFormatting = DateTimeCellFormatting();
dateTimeCellFormatting.dateListID = DatePatterns::FullDate;
dateTimeCellFormatting.timeListID = TimePatterns::ShortTime;
cell.setFormat(dateTimeCellFormatting);
std::printf("Date / time format: %d", cell.getFormat()); // 8
std::printf("Value: %d", cell.getRange().extractText().c_str()); // понедельник,
1 января 1900 г. 7 : 12

// Формат : Экспоненциальный
FractionCellFormatting fractionCellFormatting = FractionCellFormatting();
fractionCellFormatting.minNumeratorDigits = 2;
cell.setFormat(fractionCellFormatting);
std::printf("Fraction format: %d", cell.getFormat()); // 9
std::printf("Value: %d", cell.getRange().extractText().c_str()); // 2 2 / 7

// Формат : Научный
ScientificCellFormatting cellFormatting = ScientificCellFormatting();
cellFormatting.decimalPlaces = 5;
cell.setFormat(cellFormatting);
std::printf("Scientific format: %d", cell.getFormat()); // 10
std::printf("Value: %d", cell.getRange().extractText().c_str()); // 2, 30000E+00
```

6.9.5 Метод Cell::getFormat

Метод возвращает формат ячейки. Список поддерживаемых форматов ячеек приведен в разделе `CellFormat`.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
PercentageCellFormatting cellFormatting = PercentageCellFormatting();
cellFormatting.decimalPlaces = 2;
cell.setFormat(cellFormatting);
std::printf("%d", cell.getFormat());
```

6.9.6 Метод Cell::getFormattedValue

Метод позволяет получить значение ячейки в текущем формате. Список поддерживаемых форматов см. в разделе `CellFormat`.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
cell.setNumber(2.3);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.9.7 Метод Cell::setFormattedValue

Анализирует переданное значение и автоматически устанавливает формат ячейки и ее значение. В случае, если распознать тип переданного значения не удается, то для ячейки устанавливается формат `CellFormat::Text`.

Список поддерживаемых форматов см. в разделе `CellFormat`.

6.9.8 Метод Cell::unmerge

Разъединяет несколько ячеек, которые были объединены ранее.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
cell.unmerge();
```

6.9.9 Метод Cell::setContent

Определяет и устанавливает соответствующую формулу или значение, а затем форматирует ячейку. Устанавливает текст, если автоопределение не удалось.

Пример:

```
Cell cell = firstSheet.getCell("A1");
cell.setContent("=A2+A3");
```

6.9.10 Метод Cell::getBorders

Позволяет получить границы ячейки.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
Borders borders = cell.getBorders();
std::printf("%d", borders.getLeft());
```

6.9.11 Метод Cell::getRawValue

Возвращает значение ячейки в формате «Общий» (без форматирования).

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
std::printf("%s", cell.getRawValue().c_str());
```

6.9.12 Метод Cell::getCustomFormat

Возвращает строку формата ячейки.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
std::printf("%s", cell.getCustomFormat().c_str());
```

6.9.13 Метод Cell::setCustomFormat

Устанавливает формат ячейки.

Пример:

```
Cell cell = firstSheet.getCell("A1");
cell.setCustomFormat("0,00");
```

6.9.14 Метод Cell::setBool

Устанавливает для ячейки значение логического типа.

Пример:

```
Cell cell = sheet.getCell("A1");
cell.setBool(true);
```

6.9.15 Метод Cell::setNumber

Устанавливает для ячейки значение числового типа.

Пример:

```
Cell cell = sheet.getCell("A1");
cell.setNumber(0.0001);
```

6.9.16 Метод Cell::setText

Устанавливает для ячейки значение строкового типа.

Пример:

```
Cell cell = sheet.getCell("A1");
cell.setText("One");
```

6.9.17 Метод Cell::getFormulaAsString

Возвращает текст формулы ячейки. Формула – это любое выражение в ячейке, которое начинается со знака равенства (=).

Пример:

```
Cell cell = sheet.getCell("A1");
std::printf("%s", cell.getFormulaAsString().c_str());
```

6.9.18 Метод Cell::getCellProperties

Позволяет получить свойства `CellProperties` ячейки.

Пример:

```
Cell cell = sheet.getCell("A1");
CellProperties cellProperties = cell.getCellProperties();
std::printf("%d", cellProperties.verticalAlignment);
```

6.9.19 Метод Cell::setCellProperties

Позволяет установить свойства ячейки `CellProperties`.

Пример:

```
Cell cell = sheet.getCell("A1");
CellProperties cellProperties = cell.getCellProperties();
cellProperties.verticalAlignment = VerticalAlignment::Center;
cell.setCellProperties(cellProperties);
std::printf("%d", cellProperties.verticalAlignment);
```

6.9.20 Метод Cell::getParagraphProperties

Возвращает свойства абзаца `ParagraphProperties`, находящегося в ячейке.

Пример:

```
Cell cell = sheet.getCell("A1");
ParagraphProperties paragraphProperties = cell.getParagraphProperties();
if (paragraphProperties.alignment.has_value()) {
```

```
    std::printf("%d", paragraphProperties.alignment.get());
}
```

6.9.21 Метод Cell::setParagraphProperties

Устанавливает свойства абзаца `ParagraphProperties`, находящегося в ячейке.

Пример:

```
Cell cell = sheet.getCell("A1");
ParagraphProperties paragraphProperties = cell.getParagraphProperties();
paragraphProperties.alignment = Alignment::Center;
cell.setParagraphProperties(paragraphProperties);
```

6.9.22 Метод Cell::getPivotTable

Возвращает сводную таблицу `PivotTable`, относящуюся к ячейке.

Пример:

```
Table sheet = document.getBlocks().getTable(0).get();
Cell cell = sheet.getCell("A1");
boost::optional<PivotTable> pivotTable = cell.getPivotTable();
if (pivotTable.has_value()) {
    std::printf("%d", pivotTable.get().getSourceRangeAddress().c_str());
}
```

6.10 Класс CellFormat

По умолчанию при создании документа всем ячейкам присваивается формат «Общий». Полный список форматов представлен в таблице 4.

Таблица 4 – Поддерживаемые форматы ячеек таблицы

Наименование константы	Описание
`CellFormat.General`	Формат ячейки «Общий». В этом формате в ячейке отображаются первые 9 символов числа, остальные доступны для просмотра в строке формул. Для дробных чисел в формате «Общий» незначащие нули в дробной части не отображаются. Числа, состоящие более чем из 12 символов, переводятся в экспоненциальную форму после завершения ввода в ячейку.
`CellFormat.Percentage`	Формат ячейки «Процентный». Этот формат используется для представления чисел как процентов. При применении формата «Процентный»

Наименование константы	Описание
	введенное число умножается на 100 и обозначается знаком «%».
`CellFormat.Number`	Формат ячейки «Числовой». Если в ячейке с форматом «Числовой» содержится дробное число, то можно указать количество знаков, отображаемых в данном числе после разделителя.
`CellFormat.Text`	Формат ячейки «Текстовый».
`CellFormat.Currency`	Формат ячейки «Денежный». Этот формат используется для представления чисел со знаком или кодом валюты.
`CellFormat.Accounting`	Формат ячейки «Финансовый». Этот формат применяется для чисел, используемых в бухгалтерских документах. В формате «Финансовый» введенное число автоматически дополняется названием валюты, которая соответствует настройкам системы компьютера. Отрицательные числа в формате «Финансовый» заключаются в круглые скобки в ячейке, а в строке формул остаются в том виде, в котором они были введены.
`CellFormat.Date`	Формат ячейки «Дата». Этот формат автоматически присваивается числам, введенным в определенном виде, например, ДД.ММ.ГГГГ.
`CellFormat.Time`	Формат ячейки «Время». Этот формат автоматически присваивается числам, введенным в определенном виде, например, ЧЧ:ММ.
`CellFormat.Fraction`	Формат ячейки «Дробный». Этот формат используется для представления дробных чисел в виде обыкновенных дробей (то есть дробная часть заменяется на числитель и знаменатель).
`CellFormat.Scientific`	Формат ячейки «Экспоненциальный». Экспоненциальный (или научный) формат используется для представления больших чисел в короткой форме. Все введенные числа длиной более 12 символов автоматически переводятся в этот формат. В ячейке число в формате «Экспоненциальный» представлено следующим образом: – целая часть, всегда состоящая из одной цифры; – разделитель целой и дробной части;

Наименование константы	Описание
	– дробная часть, по умолчанию состоящая из двух цифр; – показатель степени числа 10 в виде **E<знак показателя степени> <показатель степени>**.
`CellFormat.Custom`	Пользовательский формат.

Использование данных констант позволяет установить выбранный формат. При этом будет использованы параметры формата по умолчанию.

Примеры использования:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("B1");
cell.setFormat(CellFormat::General);
Cell cell = firstSheet.getCell("B2");
cell.setFormat(CellFormat::Percentage);
Cell cell = firstSheet.getCell("B3");
cell.setFormat(CellFormat::Number);
```

Результат:

	A	B
1	CellFormat.General	1
2	CellFormat.Percentage	100,00%
3	CellFormat.Number	1,00

Пример форматирования ячейки также приведен в разделе, описывающем установку значений ячеек.

6.11 Класс CellPosition

Класс `CellPosition` позволяет задать координаты ячейки электронной таблицы или таблицы в составе текстового документа. Также используется для описания полей `topLeft`, `rightBottom` класса `CellRangePosition`.

Примеры:

```
Table table = document.getBlocks().getTable(0).get();
Cell cell = table.getCell(CellPosition(2, 0));
```

```
boost::optional<Table> sheet = document.getBlocks().getTable(0);
if (sheet.has_value()) {
    Charts charts = sheet.get().getCharts();
    ChartRangeInfo rangeInfo = charts.getChart(0).getRange(0);
    TableRangeInfo tableRangeInfo = rangeInfo.tableRangeInfo;
```

```
    CellRangePosition tableRange = tableRangeInfo.tableRange;
    CellPosition topLeftCellPosition = tableRange.topLeft;
    std::printf("%d %d", topLeftCellPosition.row, topLeftCellPosition.column);
}
```

6.11.1 Поле CellPosition::column

Номер столбца в значении ячейки. Нумерация столбцов начинается с нуля.

Пример:

```
CellPosition cellPosition = CellPosition();
cellPosition.column = 1;
```

6.11.2 Поле CellPosition::row

Номер строки в позиции ячейки. Нумерация строк начинается с нуля.

Пример:

```
CellPosition cellPosition = CellPosition();
cellPosition.row = 1;
```

6.11.3 Метод CellPosition::toString

Возвращает координаты ячейки в формате `(row: R, column: C)`, где `R` и `C` - номер строки и столбца соответственно.

Пример:

```
CellPosition cellPosition = CellPosition(0, 0);
std::printf("%s", cellPosition.toString().c_str());
```

6.11.4 Оператор ==

Метод используется для определения эквивалентности двух объектов `CellPosition`.

6.11.5 Оператор !=

Метод используется для определения неэквивалентности двух объектов `CellPosition`.

6.12 Класс CellProperties

Класс `CellProperties` предназначен для форматирования содержимого в ячейках таблицы. Описание полей представлено в таблице 5.

Для задания свойств ячейки используется метод `Cell::setCellProperties()`. Для получения свойств ячейки используется метод `Cell::getCellProperties()`. Иерархия классов и полей `CellProperties` отображена на рисунке 21.

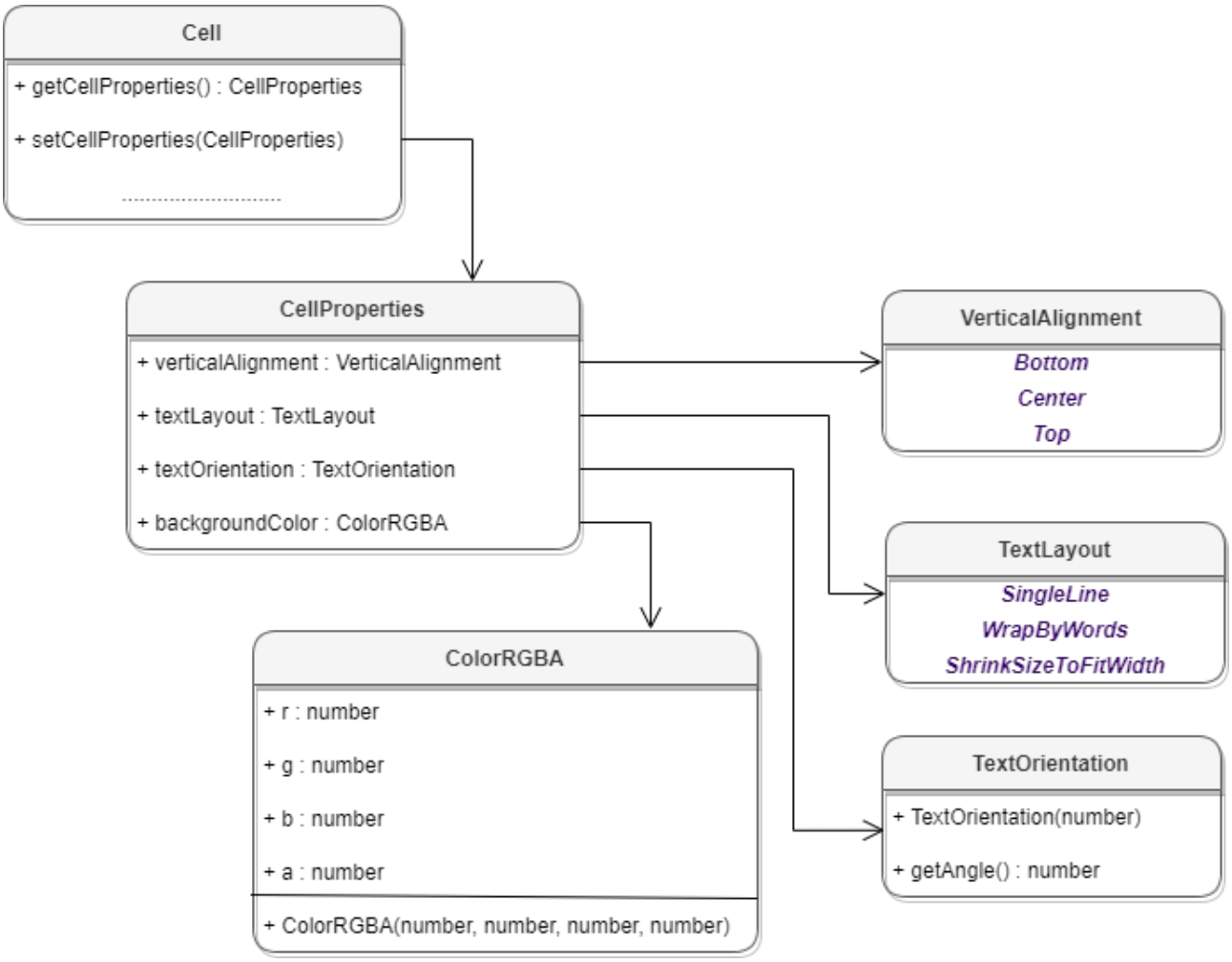


Рисунок 21 – Объектная модель для работы со свойствами ячеек таблицы

Таблица 5 – Описание полей класса `CellProperties`

Поле	Тип	Значение
`CellProperties.verticalAlignment`	`VerticalAlignment`	Вертикальное выравнивание в ячейке
`CellProperties.textLayout`	`TextLayout`	Способ отображения значения ячейки
`CellProperties.backgroundColor`	`ColorRGBA`	Цвет фона ячейки
`CellProperties.textOrientation`	`TextOrientation`	Ориентация текста в ячейке (угол поворота)

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A3");

CellProperties cellProps = cell.getCellProperties();
cellProps.verticalAlignment = VerticalAlignment::Center;
cellProps.textLayout = TextLayout::ShrinkSizeToFitWidth;
cellProps.backgroundColor = ColorRGBA(255, 255, 0, 255);
cellProps.textOrientation = TextOrientation(45);

cell.setCellProperties(cellProps);
```

6.13 Класс CellRange

Класс `CellRange` описывает диапазон ячеек таблицы.

Примеры:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
```

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRangePosition cellRangePosition = CellRangePosition(0, 0, 5, 5);
cellRange = firstSheet.getCellRange(cellRangePosition);
```

6.13.1 Метод CellRange::getEnumerator

Метод возвращает коллекцию ячеек в диапазоне.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
std::shared_ptr<Enumerator<Cell>> cellEnumerator = cellRange.getEnumerator();
while (cellEnumerator->isValid()) {
    Cell cell = cellEnumerator->getCurrent();
    std::printf("%s", cell.getFormattedValue().c_str());
    blocksEnumerator->goToNext();
}
```

6.13.2 Метод CellRange:getBeginRow

Метод возвращает индекс строки первой ячейки диапазона. Нумерация строк начинается с нуля.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
std::printf("%d", cellRange.getBeginRow());
```

6.13.3 Метод CellRange:getBeginColumn

Метод возвращает индекс столбца первой ячейки диапазона. Нумерация столбцов начинается с нуля.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
std::printf("%d", cellRange.getBeginColumn());
```

6.13.4 Метод CellRange:getLastRow

Метод возвращает индекс строки последней ячейки диапазона. Нумерация строк начинается с нуля.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
std::printf("%d", cellRange.getLastRow());
```

6.13.5 Метод CellRange:getLastColumn

Метод возвращает индекс столбца последней ячейки диапазона. Нумерация столбцов начинается с нуля.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
std::printf("%d", cellRange.getLastColumn());
```

6.13.6 Метод CellRange:setBorders

Метод предназначен для установки границ диапазона ячеек. Отдельные границы устанавливаются с помощью методов класса Borders.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
```

```
LineProperties lineProperties = LineProperties();
lineProperties.style = LineStyle::Dash;
lineProperties.width = 1.5;
lineProperties.color = Color(ColorRGBA(255, 0, 0, 255));

Borders newBorders = Borders();
newBorders.setLeft(lineProperties);
newBorders.setRight(lineProperties);
newBorders.setTop(lineProperties);
newBorders.setBottom(lineProperties);

cell.setBorders(newBorders);
```

6.13.7 Метод CellRange:insertCurrentDateTime

Метод служит для установки значения даты/времени DateTimeFormat для диапазона ячеек.

6.13.8 Метод CellRange:getCellProperties

Метод возвращает набор свойств форматирования (CellProperties) для диапазона ячеек. Возвращаемая структура содержит свойства, общие для всех ячеек диапазона.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
CellProperties cellProperties = cellRange.getCellProperties();
std::printf("%d", cellProperties.backgroundColor.get().r);
```

6.13.9 Метод CellRange:setCellProperties

Метод предназначен для установки свойств CellProperties ячеек диапазона.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
CellProperties cellProperties = CellProperties();
cellProperties.backgroundColor = ColorRGBA(55, 146, 179, 200);
cellRange.setCellProperties(cellProperties);
```

6.13.10 Метод CellRange:merge

Метод объединяет несколько ячеек таблицы в одну. Группа ячеек (диапазон) формируется с помощью класса `CellRange`. Содержимое крайней левой ячейки диапазона помещается в объединенной ячейке.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
CellRange cellRange = firstSheet.getCellRange("B3:C4");
cellRange.merge();
```

6.13.11 Метод CellRange:unmerge

Метод разъединяет ранее объединенные ячейки.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A1");
cell.unmerge();
```

6.14 Класс CellRangePosition

Класс `CellRangePosition` представляет положение диапазона ячеек в таблице. Используется в качестве поля `tableRange` класса `TableRangeInfo`, а также в методах `Table::getCellRange()`, `Chart::setRange()`. По умолчанию диапазон включает одну ячейку в позиции 0,0 что соответствует верхней левой ячейке таблицы для редактора текста, либо ячейке A1 для редактора таблиц.

Описание полей класса `CellRangePosition` представлено в таблице 6.

Таблица 6 – Поля класса `CellRangePosition`

Поле	Тип	Описание
`topLeft`	`CellPosition`	Позиция левой верхней ячейки таблицы прямоугольного диапазона. Значение 0,0 соответствует верхней левой ячейке таблицы для редактора текста, либо ячейке A1 для редактора таблиц.
`bottomRight`	`CellPosition`	Содержит позицию правой нижней ячейки таблицы прямоугольного диапазона.

Примеры:

```
Table table = document.getBlocks().getTable(0).get();
CellRangePosition cellRangePosition = CellRangePosition(0, 0, 5, 5);
CellRange range = table.getCellRange(cellRangePosition);
```

```
Table table = document.getBlocks().getTable(0).get();
Charts charts = table.getCharts();
ChartRangeInfo rangeInfo = charts.getChart(0).getRange(0);
TableRangeInfo cellRangePosition = rangeInfo.tableRangeInfo;
CellRangePosition tableRange = cellRangePosition.tableRange;
std::printf("topLeft=%d, %d", tableRange.topLeft.row,
tableRange.topLeft.column);
std::printf("bottomRight=%d, %d", tableRange.bottomRight.row,
tableRange.bottomRight.column);
```

6.14.1 Метод CellRangePosition:toString

Возвращает информацию о диапазоне ячеек в виде строкового значения формата `(topLeft: <value>, bottomRight: <value>)`.

Пример:

```
CellRangePosition cellRangePosition = CellRangePosition(0, 0, 5, 5);
std::printf("%s", cellRangePosition.toString().c_str()); // [topLeft: (row: 0,
column: 0), bottomRight: (row: 5, column: 5)]
```

6.14.2 Оператор ==

Оператор сравнения **==** используется для определения эквивалентности значений двух объектов `CellRangePosition`.

```
bool operator == (const CellRangePosition& other) const;
```

6.14.3 Оператор !=

Оператор сравнения **!=** используется для определения эквивалентности значений двух структур `CellRangePosition`.

```
bool operator != (const CellRangePosition& other) const;
```

6.15 Класс Chart

Класс `Chart` представляет диаграмму в табличном документе и описывает все ее элементы (заголовок, легенда, тип, данные, диапазон и т.д). Объектная модель класса `Chart` приведена на рисунке 22.

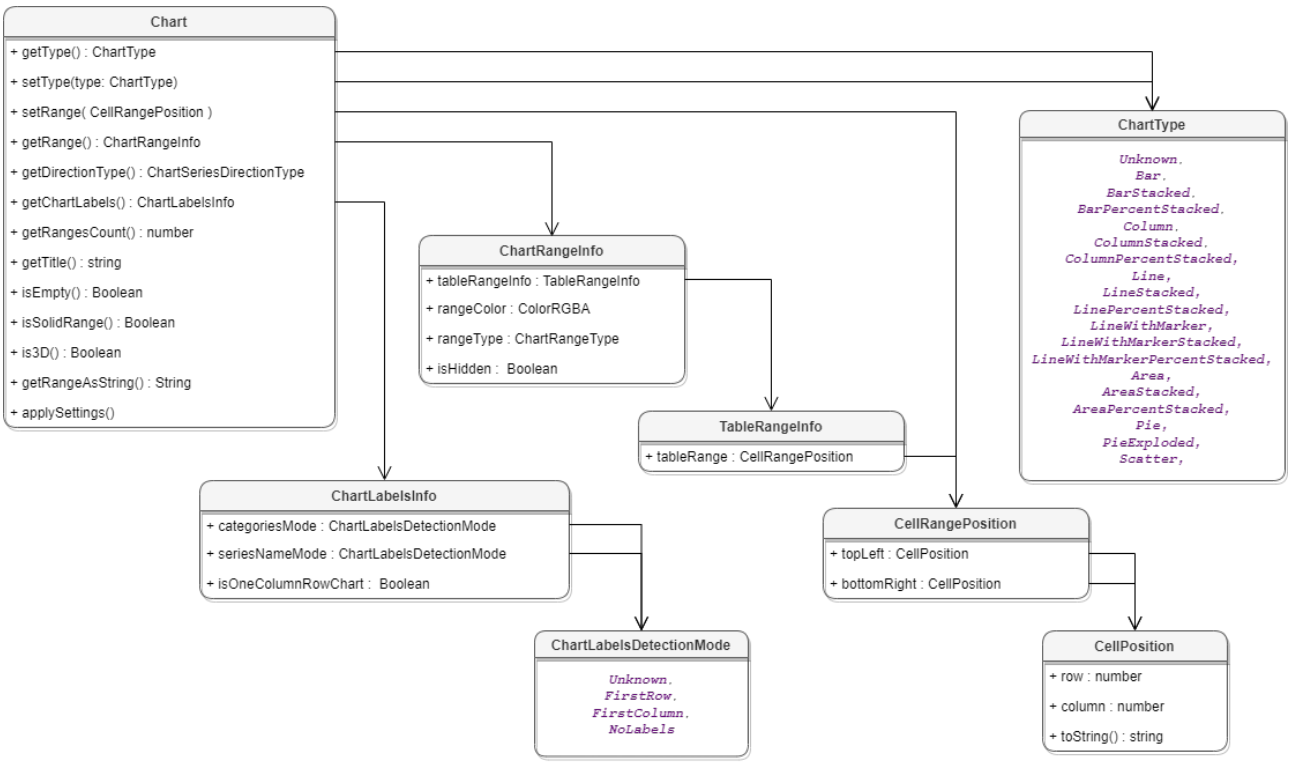


Рисунок 22 – Объектная модель класса `Chart`

6.15.1 Метод Chart::getType

Метод возвращает тип диаграммы `ChartType`.

Пример:

```
ChartType chartType = chart.getType();
std::printf("%d", chartType);
```

6.15.2 Метод Chart::setType

Метод устанавливает тип диаграммы `ChartType`. В качестве параметра передается новый тип диаграммы.

Пример:

```
Chart chart = charts.getChart(0);
chart.setType(ChartType::Area);
std::printf("%d", chart.getType());
```

6.15.3 Метод Chart::getRangesCount

Метод возвращает количество серий диаграммы.

Пример:

```
Chart chart = charts.getChart(0);
std::printf("%d", chart.getRangesCount());
```

6.15.4 Метод Chart::getRange

Метод возвращает диапазон ячеек `ChartRangeInfo` с исходными данными диаграммы. Параметр `rangesIndex` – индекс диапазона.

Пример:

```
Chart chart = charts.getChart(0);
std::printf("%d", chart.getRange(0).rangeType);
```

6.15.5 Метод Chart::getTitle

Метод возвращает заголовок диаграммы.

Пример:

```
Chart chart = charts.getChart(0);
boost::optional<std::string> title = chart.getTitle();
if (title.is_initialized()) {
    std::printf(title.value().c_str());
} else {
    std::printf("No title in chart");
}
```

6.15.6 Метод Chart::setRange

Метод задает диапазон `CellRangePosition` ячеек с исходными данными для диаграммы.

Пример:

```
Chart chart = charts.getChart(0);
CellRangePosition cellRangePosition(0, 0, 5, 5);
chart.setRange(cellRangePosition);
std::printf(chart.getRangeAsString().c_str());
```

6.15.7 Метод Chart::setRect

Метод задает область расположения диаграммы, параметр `rect` – новая область.

Внимание ! Метод устаревший (deprecated), оставлен для обратной совместимости и не рекомендован к использованию.

Пример:

```
Charts charts = firstSheet.getCharts();
Chart chart = charts.getChart(0);
chart.setRect(Rect<float>(0.0, 0.0, 100.0, 100.0));
```

6.15.8 Метод Chart::isEmpty

Метод возвращает `true`, если диаграмма не содержит значений.

Пример:

```
Chart chart = charts.getChart(0);
std::printf("%d", chart.isEmpty());
```

6.15.9 Метод Chart::isSolidRange

Метод возвращает `true`, если диапазон исходных данных диаграммы может быть выделен одним прямоугольником и не имеет промежутков.

Пример:

```
Chart chart = charts.getChart(0);
std::printf("%d", chart.isSolidRange());
```

6.15.10 Метод Chart::is3D

Метод возвращает `true`, если диаграмма трехмерная.

Пример:

```
Chart chart = charts.getChart(0);
std::printf("%d", chart.is3D());
```

6.15.11 Метод Chart::getDirectionType

Метод возвращает направление `ChartSeriesDirectionType` серий диаграммы.

Пример:

```
Chart chart = charts.getChart(0);
std::printf("%d", chart.getDirectionType());
```

6.15.12 Метод Chart::getChartLabels

Метод возвращает коллекцию меток диаграммы типа ChartLabelsInfo.

Пример:

```
Chart chart = charts.getChart(0);
ChartLabelsInfo chartInfo = chart.getChartLabels();
std::printf("%d, %d, %d", chartInfo.categoriesMode, chartInfo.seriesNameMode,
chartInfo.isOneColumnRowChart);
```

6.15.13 Метод Chart::getRangeAsString

Метод возвращает диапазон ячеек диаграммы в формате строки.

Пример:

```
Chart chart = charts.getChart(0);
std::printf(chart.getRangeAsString().c_str());
```

6.15.14 Метод Chart::applySettings

Метод позволяет обновить параметры текущей выбранной диаграммы.

Вызов:

```
applySettings(cellRange, directionType, title, labelsInfo)
```

Параметры:

– `cellRange` – обновленный диапазон исходных данных диаграммы CellRange;

– `directionType` – направление серий ChartSeriesDirectionType;

– `title` – заголовок диаграммы (тип - строка);

– `labelsInfo` – информация о метках диаграммы ChartLabelsInfo.

Пример:

```
CellRange cellRange = firstSheet.getCellRange("B3:C4");
ChartLabelsInfo chartLabelsInfo =
ChartLabelsInfo(ChartLabelsDetectionMode::FirstColumn,
ChartLabelsDetectionMode::FirstRow, false);
chart.applySettings(cellRange, boost::none,
boost::optional<std::string>("Title"), chartLabelsInfo);
```

6.16 Класс Charts

Класс `Charts` обеспечивает доступ к списку диаграмм (см. Рисунок 23) табличного документа. Доступ к списку диаграмм осуществляется с помощью метода

Table::getCharts().

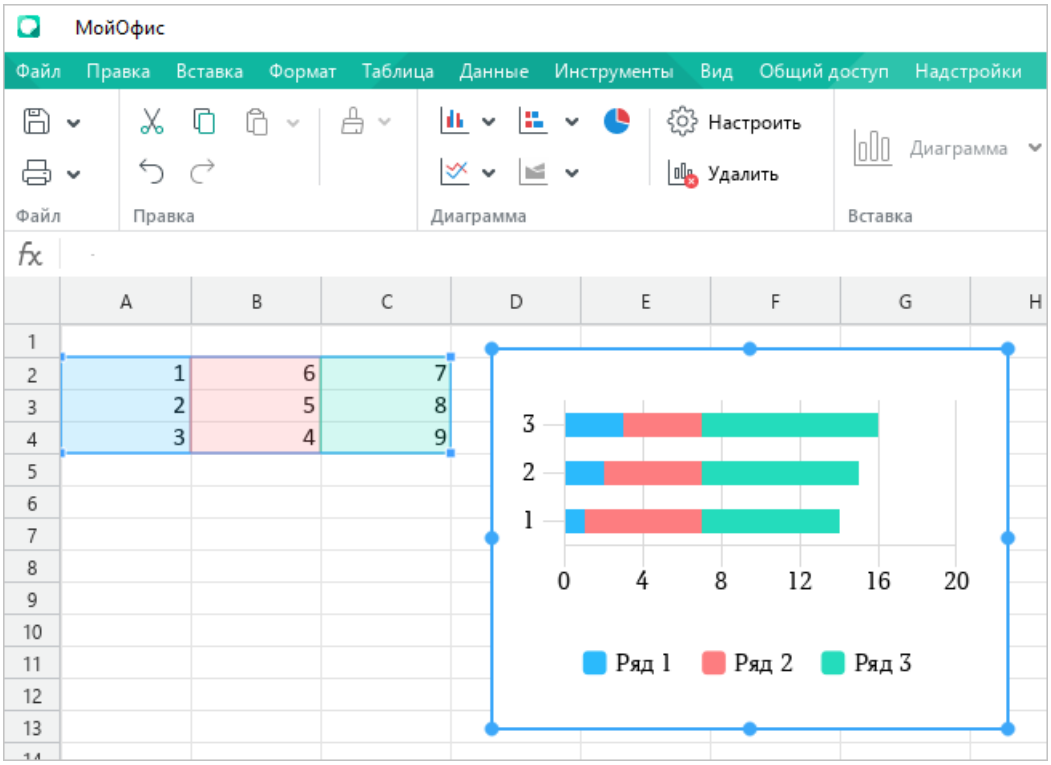


Рисунок 23 – Пример отображения диаграммы в МойОфис Таблица.

6.16.1 Метод Charts::getChartsCount

Метод возвращает общее количество диаграмм в табличном документе.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Charts charts = firstSheet.getCharts();
std::printf("%d", charts.getChartsCount());
```

6.16.2 Метод Charts::getChart

Метод возвращает диаграмму Chart по индексу `chartIndex` в коллекции диаграмм.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Charts charts = firstSheet.getCharts();
Chart chart = charts.getChart(0);
std::printf(chart.getRangeAsString());
```

6.16.3 Метод Charts::getChartIndexByDrawingIndex

Метод возвращает индекс диаграммы по индексу отрисовки `drawingIndex`.

Пример:

```
boost::optional<size_t> indexByDrawing = charts.getChartIndexByDrawingIndex(0);
if (indexByDrawing.has_value()) {
    std::printf("%d", indexByDrawing.value());
} else {
    std::printf("No index by drawing");
}
```

6.17 Класс ChartLabelsDetectionMode

Класс описывает режимы автоматического определения меток диаграмм.

```
enum class ChartLabelsDetectionMode : uint8_t
{
    Unknown = 0,
    FirstRow,
    FirstColumn,
    NoLabels
};
```

Поля класса соответвуют следующим режимам автоматического определения меток диаграмм:

- `Unknown` – неопределенный тип;
- `FirstRow` – метка на первой строке;
- `FirstColumn` – метка на первой колонке;
- `NoLabels` – не отрисовывать метки.

Пример:

```
local tbl = document:getBlocks():getTable(0)
local charts = tbl:getCharts()
local chart = charts:getChart(0)
local chartLabels = chart:getChartLabels()
print(chartLabels.categoriesMode, chartLabels.seriesNameMode)
```

6.18 Класс ChartLabelsInfo

Класс `ChartLabelsInfo` описывает настройки автоматического определения меток диаграммы. Инициализируется конструктором:


```
ChartLabelsInfo(const ChartLabelsDetectionMode categoriesMode,
                const ChartLabelsDetectionMode seriesNameMode,
                const bool oneColumnRow)
```

Параметры конструктора:

- `categoriesMode` - режим автоматического определения меток для категорий, тип `ChartLabelsDetectionMode`;
- `seriesNameMode` - режим автоматического определения меток для серий, тип `ChartLabelsDetectionMode`;
- `oneColumnRow` - передается `true`, если диапазон диаграммы содержит только одну строку или одну колонку.

Описание полей класса `ChartLabelsInfo` представлено в таблице 7.

Таблица 7 – Описание полей класса `ChartLabelsInfo`

Поле	Описание	Тип
`categoriesMode`	Режим автоматического определения меток для категорий	`ChartLabelsDetectionMode`
`seriesNameMode`	Режим автоматического определения меток для серий	`ChartLabelsDetectionMode`
`isOneColumnRowChart`	Поле содержит `true`, если диапазон диаграммы содержит только одну строку или одну колонку	Boolean

Примеры:

```
ChartLabelsInfo chartInfo(ChartLabelsDetectionMode::FirstRow,
ChartLabelsDetectionMode::NoLabels, false);
```

```
Table firstSheet = document.getBlocks().getTable(0).get();
Charts charts = firstSheet.getCharts();
Chart chart = charts.getChart(0);
ChartLabelsInfo chartInfo = chart.getChartLabels();
std::printf("%d, %d, %d", chartInfo.categoriesMode, chartInfo.seriesNameMode,
chartInfo.isOneColumnRowChart);
```

6.19 Класс ChartRangeInfo

Класс `ChartRangeInfo` описывает серию диаграммы. Инициализируется конструктором:

```
ChartRangeInfo(const CellRange& cellRange,
               const ColorRGBA& color,
               const bool hidden,
               const ChartRangeType type);
```

Параметры конструктора:

– `tableRangeInfo` - диапазон ячеек, тип `TableRangeInfo`;

– `color` – цвет серии диаграммы, тип `ColorRGBA`;

– `hidden` – видимость серии, тип Boolean;

– `rangeType` – тип диапазона исходных данных диаграммы, тип `ChartRangeType`.

Описание полей класса представлено в таблице 8.

Таблица 8 – Описание полей класса `ChartRangeInfo`

Поле	Описание	Тип
`tableRangeInfo`	Исходный диапазон ячеек для серии	`TableRangeInfo`
`rangeColor`	Цвет для отрисовки серии	`ColorRGBA`
`isHidden`	Задает видимость серии диаграммы	`bool`
`rangeType`	Тип диапазона диаграммы	`ChartRangeType`

Пример:

```
Chart chart = charts.getChart(0);
ChartRangeInfo chartRangeInfo = chart.getRange(0);
std::printf("%s, %d-%d-%d, %d, %d",
            chartRangeInfo.tableRangeInfo.tableRange.toString().c_str(),
            chartRangeInfo.rangeColor.a, chartRangeInfo.rangeColor.b,
chartRangeInfo.rangeColor.g,
            chartRangeInfo.isHidden, chartRangeInfo.rangeType);
```

6.20 Класс ChartRangeType

Класс описывает тип диапазона исходных данных диаграммы.

```
enum class ChartRangeType : uint8_t
{
    Series,
    SeriesName,
    Categories,
```

```
    DataPoint
};
```

Возможные значения:

- `Series` – серии;
- `SeriesName` – имена серий;
- `Categories` – области;
- `DataPoint` – разметка данных.

Пример:

```
Charts charts = firstSheet.getCharts();
ChartRangeInfo rangeInfo = charts.getChart(0).getRange(0);
std::string rangeTypes[] = { "Series", "SeriesName", "Categories",
"DataPoint" };
std::printf((rangeTypes[(int)rangeInfo.rangeType]).c_str());
```

6.21 Класс ChartSeriesDirectionType

Класс описывает направление серий диаграмм.

```
enum class ChartSeriesDirectionType : uint8_t
{
    Unknown,
    ByRow,
    ByColumn
};
```

Возможные значения:

- `Unknown` – неопределенный тип;
- `ByRow` – серии направлены по строкам;
- `ByColumn` – серии направлены по колонкам.

Пример:

```
ChartSeriesDirectionType directionType = charts.getChart(0).getDirectionType();
std::string chartDirectionTypes[] = { "Unknown", "ByRow", "ByColumn" };
std::printf((chartDirectionTypes[(int)directionType]).c_str());
```

6.22 Класс ChartType

Перечисление `ChartType` описывает все поддерживаемые типы диаграмм.

```
enum class ChartType : std::uint8_t
{
    Unknown = 0,
    Bar,
    BarStacked,
    BarPercentStacked,
    Column,
    ColumnStacked,
    ColumnPercentStacked,
    Line,
    LineStacked,
    LinePercentStacked,
    LineWithMarker,
    LineWithMarkerStacked,
    LineWithMarkerPercentStacked,
    Area,
    AreaStacked,
    AreaPercentStacked,
    Pie,
    PieExploded,
    Scatter
};
```

Поля класса соответвуют следующим типам диаграмм:

- `Unknown` – неопределенный тип;
- `Bar` – линейчатая диаграмма с группировкой;
- `BarStacked` – линейчатая диаграмма с накоплением;
- `BarPercentStacked` – линейчатая нормированная диаграмма с накоплением;
- `Column` – гистограмма с группировкой;
- `ColumnStacked` – гистограмма с накоплением;
- `ColumnPercentStacked` – нормированная гистограмма с накоплением;
- `Line` – стандартный график;
- `LineStacked` – график с накоплением;
- `LinePercentStacked` – нормированный график с накоплением;
- `LineWithMarker` – стандартный график с маркерами;

- `LineWithMarkerStacked` – график с накоплением и маркерами;
- `LineWithMarkerPercentStacked` – нормированный график с накоплением и маркерами;
- `Area` – стандартная диаграмма с областями;
- `AreaStacked` – диаграмма с областями с накоплением;
- `AreaPercentStacked` – нормированная диаграмма с областями с накоплением;
- `Pie` – круговая диаграмма;
- `PieExploded` – круговая диаграмма с отделенными секторами;
- `Scatter` – диаграмма рассеяния.

Пример:

```
ChartType chartType = chart.getType();
std::printf("%d", chartType);
```

6.23 Класс Color

Класс `Color` представляет либо цветовой объект RGBA, либо заданные цвета идентификатора темы. В качестве параметров конструктора используются объекты `ColorRGBA`, `ThemeColorID`.

Пример:

```
Color rgbaColor = Color(ColorRGBA(255, 0, 0, 255));
Color themeColor = Color(ThemeColorID_Text);
```

6.23.1 Метод Color::getRGBAColor

Метод возвращает цвет `ColorRGBA`.

Пример:

```
Color color = Color(ColorRGBA(255, 0, 0, 255));
const boost::optional<ColorRGBA> rgbaColor = color.getRGBAColor();
if (rgbaColor.has_value()) {
    std::printf("%d", rgbaColor.get().r);
}
```

6.23.2 Метод Color::getThemeColorID

Метод возвращает цвет идентификатора темы `ThemeColorID`.

Пример:

```
Color color = Color(ColorRGBA(255, 0, 0, 255));
const boost::optional<ThemeColorID> themeColorId = color.getThemeColorID();
if (themeColorId.has_value()) {
    std::printf("%d", themeColorId.get());
}
```

6.23.3 Оператор ==

Метод используется для определения эквивалентности двух значений цвета.

Пример:

```
Color color1 = Color(ColorRGBA(255, 0, 0, 255));
Color color2 = Color(ColorRGBA(255, 0, 0, 255));
if (color1 == color2) {
    std::printf("%s", "Colors are equal");
} else {
    std::printf("%s", "Colors are not equal");
}
```

6.23.4 Оператор !=

Метод используется для определения неэквивалентности двух значений цвета.

Пример:

```
Color color1 = Color(ColorRGBA(255, 0, 0, 254));
Color color2 = Color(ColorRGBA(255, 0, 0, 255));
if (color1 == color2) {
    std::printf("%s", "Colors are equal");
} else {
    std::printf("%s", "Colors are not equal");
}
```

6.24 Класс ColorRGBA

Класс `ColorRGBA` предназначен для задания цвета текста, линии, фона и т.д. Используется четырехканальный формат, содержащий данные для красного (`r`), зеленого (`g`), голубого (`b`) цветов и альфа-канала (`a`).

Для создания нового объекта используется один из конструкторов:

```
ColorRGBA()
ColorRGBA(uint8_t r, uint8_t g, uint8_t b, uint8_t a)
```

Описание полей класса `ColorRGBA` представлено в таблице 9.

Таблица 9 – Описание полей класса `ColorRGBA`

Поле	Тип	Описание
r	`uint8_t`	Значение от 0 до 255 для установки интенсивности красного цвета.
g	`uint8_t`	Значение от 0 до 255 для установки интенсивности зеленого цвета.
b	`uint8_t`	Значение от 0 до 255 для установки интенсивности голубого цвета.
a	`uint8_t`	Значение от 0 до 255 для регулировки прозрачности. Значение 255 соответствует полностью непрозрачному цвету.

Примеры использования:

```
ColorRGBA rgba = ColorRGBA();
rgba.r = 0;
rgba.g = 0;
rgba.b = 255;
rgba.a = 200;
// r: 0, g: 0, b: 255, a: 200
std::printf("r: %d, g: %d, b: %d, a: %d", rgba.r, rgba.g, rgba.b, rgba.a);
```

```
rgba = ColorRGBA(55, 146, 179, 200);
// r: 55, g: 146, b: 179, a: 200
std::printf("r: %d, g: %d, b: %d, a: %d", rgba.r, rgba.g, rgba.b, rgba.a);
```

```
LineProperties lineProps = LineProperties();
lineProps.color = Color(rgba);
```

6.24.1 Оператор ==

Метод используется для определения эквивалентности двух объектов `ColorRGBA`.

Пример:

```
ColorRGBA colorRGBA_First = ColorRGBA(255, 255, 255, 0);
ColorRGBA colorRGBA_Second = ColorRGBA(255, 255, 255, 0);
if (colorRGBA_First == colorRGBA_Second) {
    // equals
}
```

6.24.2 Оператор !=

Метод используется для определения неэквивалентности двух двух объектов `ColorRGBA`.

Пример:

```
ColorRGBA colorRGBA_First = ColorRGBA(255, 255, 255, 0);
ColorRGBA colorRGBA_Second = ColorRGBA(255, 255, 255, 1);
if (colorRGBA_First != colorRGBA_Second) {
    // not equals
}
```

6.25 Класс Comment

Класс `Comment` предоставляет доступ к следующим свойствам комментария:

- диапазон текста `Range`, который описывает комментарий;
- текст комментария;
- информация о комментарии `TrackedChangeInfo`;
- признак того, что комментарий принят;
- список ответов на комментарий `Comments`.

6.25.1 Метод Comment::getRange

Метод возвращает диапазон документа `Range`, которому соотвествует комментарий.

Пример:

```
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
if (enumerator) {
    while (enumerator->isValid()) {
        Comment comment = enumerator->getCurrent();
        Range commentRange = comment.getRange();
        std::printf("Text: %s", commentRange.extractText().c_str());
        enumerator->goToNext();
     }
}
```

6.25.2 Метод Comment::getText

Метод возвращает текст комментария.

Пример:

```
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
if (enumerator) {
    while (enumerator->isValid()) {
```

```
        Comment comment = enumerator->getCurrent();
        boost::optional<std::string> commentText = comment.getText();
        if (commentText.has_value()) {
            std::printf("Text: %s", commentText.get().c_str());
        }
        enumerator->goToNext();
    }
}
```

6.25.3 Метод Comment::getInfo

Метод предоставляет доступ к информации о комментарии TrackedChangeInfo (автор изменения, дата и т. д).

Пример:

```
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
if (enumerator) {
    while (enumerator->isValid())
    {
        Comment comment = enumerator->getCurrent();
        TrackedChangeInfo info = comment.getInfo();
        boost::optional<UserInfo> authorOpt = info.author;
        if (authorOpt.has_value()) {
            std::printf("Author: %s\n", authorOpt.get().name.c_str());
        }
        enumerator->goToNext();
    }
}
```

6.25.4 Метод Comment::isResolved

Метод возвращает значение `true`, если комментарий принят.

Пример:

```
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
if (enumerator) {
    while (enumerator->isValid()) {
        Comment comment = enumerator->getCurrent();
        std::printf("Resolved: %d", comment.isResolved());
```

```
        enumerator->goToNext();
    }
}
```

6.25.5 Метод Comment::getReplies

Метод предоставляет доступ к ответам на комментарии. Ответы находятся в таком же классе `Comments`, как и сами комментарии документа.

Пример:

```
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
if (enumerator) {
    while (enumerator->isValid()) {
        Comment comment = enumerator->getCurrent();
        std::shared_ptr<Enumerator<Comment>> replies =
comment.getReplies().getEnumerator();
        if (replies) {
            while (replies->isValid()) {
                boost::optional<std::string> replyTextOpt = replies.get()-
>getCurrent().getText();
                std::printf("%s", replyTextOpt.get().c_str());
                replies->goToNext();
            }
        }
        enumerator->goToNext();
    }
}
```

6.26 Класс Comments

Класс `Comments` содержит коллекцию комментариев диапазона (см. Рисунок 24).

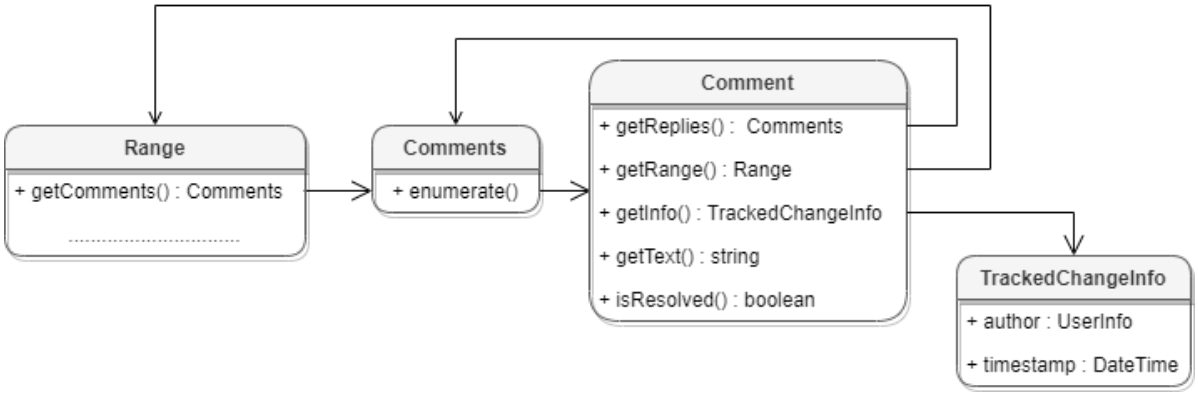


Рисунок 24 – Объектная модель классов для работы с комментариями

Для получения списка комментариев используется метод `Range::getComments()`.

Пример:

```
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
if (enumerator) {
    while (enumerator->isValid()) {
        Comment comment = enumerator->getCurrent();
        ..........................
        enumerator->goToNext();
    }
}
```

6.26.1 Метод Comments::getEnumerator

Метод возвращает коллекцию комментариев всего документа.

Пример:

```
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
if (enumerator) {
    while (enumerator->isValid()) {
        ......................
        enumerator->goToNext();
    }
}
```

6.27 Класс Connection

Класс `Connection` реализует соединение между `Messenger` и клиентом. Содержит один метод `unsubscribe` для разрыва соединения.

Пример:

```
Messenger::MessageHandlerFunction handler;
std::shared_ptr<Messenger> messenger = application.getMessenger();
std::shared_ptr<Connection> connection = messenger->subscribe(handler);
```

6.28 Класс CurrencyCellFormatting

Класс содержит параметры для денежного формата ячеек таблицы. Описание полей класса `CurrencyCellFormatting` представлено в таблице 10.

Таблица 10 – Описание полей класса `CurrencyCellFormatting`

Поле	Описание
`CurrencyCellFormatting.decimalPlaces`	Количество десятичных позиций
`CurrencyCellFormatting.symbol`	Символ денежной единицы
`CurrencyCellFormatting.localeCode`	Идентификатор кода языка (MS-LCID)
`CurrencyCellFormatting.useThousandsSeparator`	Использовать разделитель для тысячных
`CurrencyCellFormatting.useRedForNegative`	Использовать красный цвет для отрицательных значений
`CurrencyCellFormatting.useBracketsForNegative`	Использовать скобки для отрицательных значений
`CurrencyCellFormatting.hideSign`	Скрывать знак «минус» для отрицательных значений
`CurrencyCellFormatting.currencySignPlacement`	Варианты размещения знака валюты `CurrencySignPlacement`

Экземпляр данного класса используется в качестве аргумента метода `Cell::setFormat()`, см. пример.

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("A2");

CurrencyCellFormatting cellFormat = CurrencyCellFormatting();
```

```
cellFormat.decimalPlaces = 2;
cellFormat.useThousandsSeparator = true;
cellFormat.useRedForNegative = true;
cellFormat.useBracketsForNegative = true;
cellFormat.hideSign = false;
cellFormat.currencySignPlacement = CurrencySignPlacement::Suffix;

cell.setFormat(cellFormat);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.29 Класс CurrencySignPlacement

Варианты размещения знака валюты представлены в таблице 11. Данный тип используется в поле `currencyFormat` класса `LocaleInfo`, а также в поле `currencySignPlacement` класса `CurrencyCellFormatting` (см.пример в ее описании).

Таблица 11 – Описание полей класса `CurrencySignPlacement`

Поле	Описание	Пример
`CurrencySignPlacement::Prefix`	Размещение знака валюты до значения	$12.00
`AccountingCellFormatting::Suffix`	Размещение знака валюты после значения	12,00 ₽

6.30 Класс DatePatterns

Форматы даты представлены в таблице 12. Пример использования см. в главе `DateTimeCellFormatting`.

Таблица 12 – Форматы даты

Наименование константы	Описание
`DatePatterns::DayMonthTextLongYearLong`	`'mmmm dd, yyyy'` для языка `en_US`
`DatePatterns::FullDate`	`'день недели, mmmm dd, yyyy'` для языка `en_US`
`DatePatterns::DayMonthNumberLongYearLong`	`'mm/dd/yyyy'` для языка `en_US`
`DatePatterns::DayMonthNumberLongYearShort`	`'m/dd/yy'` для языка `en_US`
`DatePatterns::DayMonthNumberShortYearShort`	`'dd-mmm'` для языка `en_US`
`DatePatterns::DayMonthTextShort`	`'mmm-yy'` для языка `en_US`
`DatePatterns::DayMonthTextShortYearShort`	`'mmm dd, yy'` для языка `en_US`

Наименование константы	Описание
DatePatterns::DayMonthYearLongNonLocalizableHyphen	Нелокализуемый шаблон 'dd-mm-yyyy'
DatePatterns::DayMonthYearLongNonLocalizableSlash	Нелокализуемый шаблон 'dd/mm/yyyy'

6.31 Класс DateTime

Класс CO:API:DateTime предоставляет дату и время с точностью до секунды. Используется для поля TrackedChangeInfo.timeStamp. Описание полей класса DateTime представлено в таблице 13.

Таблица 13 – Описание полей класса DateTime

Поле	Тип	Описание
DateTime.year	Дата	Год
DateTime.month	Дата	Месяц
DateTime.day	Дата	День
DateTime.hour	Время	Часы
DateTime.minute	Время	Минуты
DateTime.second	Время	Секунды

6.31.1 Оператор ==

Оператор сравнения == используется для определения эквивалентности значений двух структур DateTime.

```
bool operator == (const DateTime& other) const;
```

6.31.2 Оператор !=

Оператор сравнения != используется для определения неэквивалентности значений двух структур DateTime.

```
bool operator != (const DateTime& other) const;
```

6.32 Класс DateTimeCellFormatting

Класс содержит параметры для формата ячеек таблицы типа Дата и Время, используется в качестве аргумента метода Cell::setFormat(). Описание полей класса DateTimeCellFormatting представлено в таблице 14.

Таблица 14 – Описание полей класса `DateTimeCellFormatting`

Поле	Описание
`DateTimeCellFormatting.dateListID`	Формат даты `DatePatterns`
`DateTimeCellFormatting.timeListID`	Формат времени `TimePatterns`

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("A2");

DateTimeCellFormatting cellFormat = DateTimeCellFormatting();
cellFormat.dateListID = DatePatterns::DayMonthNumberLongYearShort;
cellFormat.timeListID = TimePatterns::LongTime;

cell.setFormat(cellFormat);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.33 Класс DateTimeFormat

В таблице 15 представлены варианты масштабирования при печати табличных документов. Используется в качестве параметра метода `CellRange::insertCurrentDateTime()`.

Таблица 15 – Варианты масштабирования при печати табличных документов

Наименование константы	Описание
`DateTimeFormat::DateTime`	Дата/время
`DateTimeFormat::Date`	Дата
`DateTimeFormat::Time`	Время

6.34 Класс Document

Класс `Document` осуществляет доступ к содержимому открытого текстового или табличного документа.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    ..........................
}
```

6.34.1 Метод Document::saveAs

Метод `Document::saveAs` сохраняет документ в файл по указанному пути. Формат и тип документа определяются расширением файла, если они не указаны в явном виде.

При необходимости, в качестве второго аргумента можно использовать объект класса `SaveDocumentSettings`, которая содержит формат документа `DocumentFormat`, тип документа `DocumentType` и пароль для защиты документа от несанкционированного доступа.

Примеры:

```
document.saveAs(filePath);
```

```
SaveDocumentSettings saveDocumentSettings = SaveDocumentSettings();

saveDocumentSettings.documentFormat = DocumentFormat::OXML;
saveDocumentSettings.documentType = DocumentType::Workbook;
saveDocumentSettings.documentPassword = "password";
saveDocumentSettings.isTemplate = false;

saveDocumentSettings.dsvSettings = DSVSettings();
saveDocumentSettings.dsvSettings.autofit = true;
saveDocumentSettings.dsvSettings.startBlockIndex = 0;
saveDocumentSettings.dsvSettings.lastBlockIndex = 10;

document.saveAs(filePath, saveDocumentSettings);
```

6.34.2 Метод Document::exportAs

Метод `Document::exportAs` экспортирует документ в файл по указанному пути с указанным форматом.

Расширенные версии метода позволяют указать дополнительные настройки экспорта документа:

- для текстовых документов – класс `TextExportSettings`;
- для табличных документов – класс `WorkbookExportSettings`.

В настоящее время поддерживается только операция экспорта документа в формат PDF.

Примеры:

```
document.exportAs(filePath, ExportFormat::PDFA1)
```

```
TextExportSettings textExportSettings = TextExportSettings();
textExportSettings.pageNumbers = PageNumbers(PageParity::Even);
document.exportAs(filePath, ExportFormat::PDFA1, textExportSettings);
```

```
WorkbookExportSettings workbookSettings = WorkbookExportSettings();
workbookSettings.sheetNames = SheetNames();
workbookSettings.sheetNames.push_back("Лист2");
workbookSettings.printingScope = PrintingScope(PrintingScope::Type::PrintArea);
workbookSettings.pageProperties = PageProperties(100, 200);
workbookSettings.scale = 90;
document.exportAs(filePath, ExportFormat::PDFA1, workbookSettings);
```

6.34.3 Метод Document::merge

Метод `Document::merge` сравнивает текущий документ с другим документом, который передается в параметре типа `Document`.

Метод возвращает объект `Document`, содержащий результат сравнения в виде отслеживаемых изменений.

Пример:

```
CO::API::Application application;

CO::API::Document::Document firstDoc =
application.loadDocument("C:/Tmp/Sample1.docx");
CO::API::Document::Document secondDoc =
application.loadDocument("C:/Tmp/Sample2.docx");

CO::API::Document::Document mergedDoc = firstDoc.merge(secondDoc);
auto outputFilePath = "C:/Tmp/BasicExample3.docx";
mergedDoc.saveAs(outputFilePath);
```

Результат выполнения данного примера (сравнение двух документов, содержащих "1111" и "2222") приведен на рисунке 25.

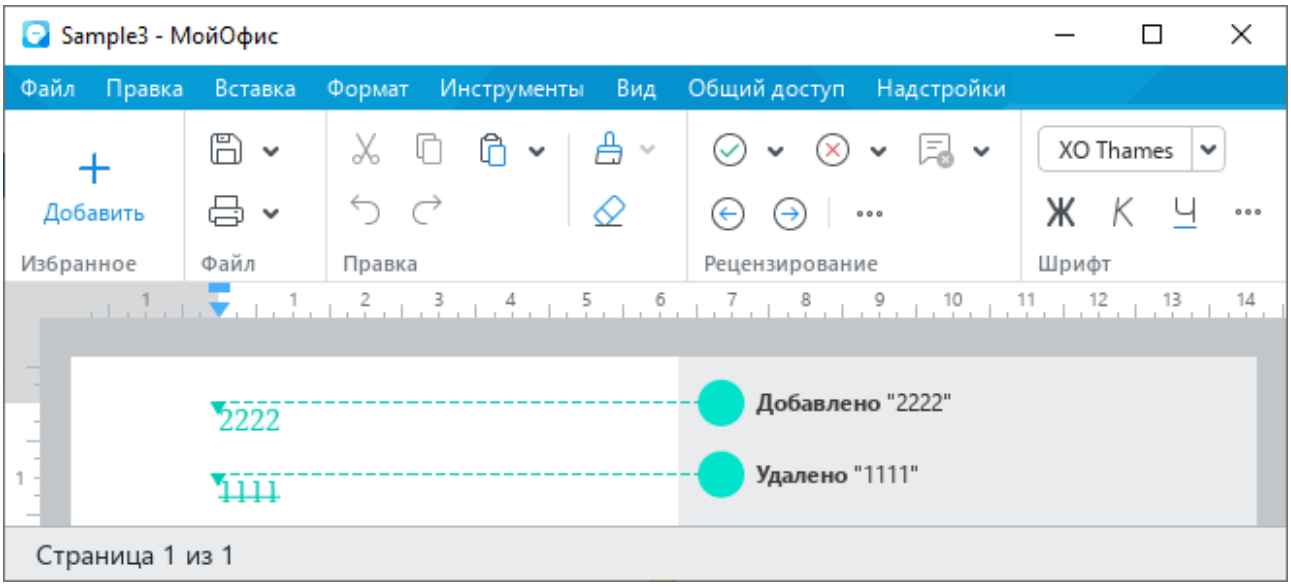


Рисунок 25 – Результат выполнения метода `merge`

6.34.4 Метод Document::getBlocks

Метод предоставляет доступ к объекту `Blocks` и далее к отдельным фрагментам (абзацам, таблицам и т. д.), из которых состоит документ.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
```

6.34.5 Метод Document::getBookmarks

Метод предоставляет доступ к списку закладок `Bookmarks`.

Пример:

```
Bookmarks bookmarks = document.getBookmarks();
boost::optional<Range> bookmarkRange = bookmarks.getBookmarkRange("Boomark");
if (bookmarkRange.has_value()) {
    bookmarkRange.get().replaceText("New bookmark text");
    std::printf("Bookmark range text : %s",
bookmarkRange.get().extractText().c_str());
}
```

6.34.6 Метод Document::getScripts

Метод предоставляет доступ к списку макрокоманд `Scripts`, содержащихся в документе.

Пример:

```
std::shared_ptr<Scripts> scripts = document.getScripts();
if (scripts) {
    std::shared_ptr<Enumerator<std::shared_ptr<Script>>> enumerator = scripts-
>getEnumerator();
    while (enumerator->isValid()) {
        enumerator->goToNext();
    }
}
```

6.34.7 Метод Document::getRange

Метод предоставляет доступ ко всему диапазону Range документа.

Пример:

```
Range range = document.getRange();
std::printf("%s", range.extractText().c_str());
```

6.34.8 Метод Document::isChangesTrackingEnabled

Метод возвращает текущее состояние отслеживания изменений в документе (`true` - включены).

Пример:

```
std::printf("%d", document.isChangesTrackingEnabled());
```

6.34.9 Метод Document::setChangesTrackingEnabled

Метод управляет состоянием отслеживания изменений в документе (включены или выключены).

Пример:

```
document.setChangesTrackingEnabled(true);
```

6.34.10 Метод Document::getComments

Метод обеспечивает доступ к комментариям документа, возвращает объект Comments.

Пример:

```
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
if (enumerator) {
    while (enumerator->isValid()) {
        Comment comment = enumerator->getCurrent();
```

```
        ..........................
        enumerator->goToNext();
    }
}
```

6.34.11 Метод Document::setPageProperties

Метод устанавливает свойство PageProperties в документе.

Пример:

```
PageProperties pageProperties = PageProperties();
pageProperties.width = 100;
pageProperties.height = 200;
document.setPageProperties(pageProperties);
```

6.34.12 Метод Document::setPageOrientation

Метод устанавливает ориентацию страниц в документе (см. PageOrientation).

Пример:

```
document.setPageOrientation(PageOrientation::Landscape);
```

6.34.13 Метод Document::getSectionsEnumerator

Реализует доступ к объектам типа Section.

Пример:

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    ......................
    enumerator->goToNext();
}
```

6.34.14 Метод Document::getSections

Возвращает объект типа Sections.

Пример:

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    ......................
    enumerator->goToNext();
}
```

6.34.15 Метод Document::setMirroredMarginsEnabled

Метод позволяет включать и выключать зеркальные поля в документе.

Пример:

```
document.setMirroredMarginsEnabled(true);
```

6.34.16 Метод Document::areMirroredMarginsEnabled

Возвращает состояние режима зеркальных полей в документе (разрешены или запрещены).

Пример:

```
std::printf("%d", document.areMirroredMarginsEnabled());
```

6.34.17 Метод Document::getPivotTablesManager

Возвращает объект PivotTablesManager, который используется для создания сводных таблиц. Метод может быть использован только в табличном редакторе.

Пример:

```
boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
if (pivotTablesManagerOpt.has_value()) {
    PivotTablesManager pivotTablesManager = pivotTablesManagerOpt.get();
    ..................................
}
```

6.34.18 Метод Document::getNamedExpressions

Используется для получения списка именованных диапазонов NamedExpressions.

Пример:

```
NamedExpressions namedExpressions = document.getNamedExpressions();
```

6.35 Класс DocumentFormat

В таблице 16 приведены поддерживаемые форматы документов, структура используется в DocumentSettings.

Таблица 16 – Форматы документов

Наименование константы	Описание
DocumentFormat::PlainText	Используется для работы с файлами TXT.

Наименование константы	Описание
`DocumentFormat::DSV`	Используется для работы с табличными данными в текстовой форме (CSV, DSV). Строка текста содержит одно или несколько полей данных, разделенных запятыми или иным разделителем.
`DocumentFormat::OXML`	Используется для работы с текстовыми (DOCX) или табличными (XLSX) документами в формате Open Office XML.
`DocumentFormat::ODF`	Используется для работы с текстовыми (ODT) или табличными (ODS) документами формата Open Document Format (ГОСТ Р ИСО/МЭК 26300-2010).
`DocumentFormat::HTML`	Используется для работы с веб-документами в формате HTML. Работа с веб-документами в формате HTML средствами Document API в настоящий момент не поддерживается.
`DocumentFormat::PDF`	Используется для работы с документами в формате Portable Document Format (PDF) версии 1.4.
`DocumentFormat::PDFA`	Используется для работы с документами в формате Portable Document Format (PDF) для долгосрочного архивного хранения (PDF/A-1b).

6.36 Класс DocumentSettings

Класс `DocumentSettings` предоставляет общие настройки документа и используется в `document::createDocument()`.

Описание полей класса `DocumentSettings` представлено в таблице 17.

Таблица 17 – Описание полей класса `DocumentSettings`

Поле	Тип	Описание
`DocumentSettings.documentType`	`DocumentType`	Тип документа
`DocumentSettings.userInfo`	`UserInfo`	Информация о пользователе.
`DocumentSettings.localeInfo`	`LocaleInfo`	Информация о локализации
`DocumentSettings.timeZone`	`TimeZone`	Информация о временной зоне
`DocumentSettings.formulaType`	`FormulaType`	Система адресации ячеек

6.37 Класс DocumentType

В таблице 18 приведены поддерживаемые типы документов, используются при создании документа `application::createDocument()`, `DocumentSettings`.

Таблица 18 - Типы документов

Наименование константы	Описание
`DocumentType::Text`	Используется для работы с текстовыми документами в форматах DOCX, ODT, XODT, TXT.

Наименование константы	Описание
`DocumentType::Workbook`	Используется для работы с табличными документами в форматах XLSX, ODS, XODS.
`DocumentType::Presentation`	Используется для работы с презентационными документами в форматах PPTX, ODP. Работа с презентационными документами средствами Document API в настоящий момент не поддерживается.

6.38 Класс DSVSettings

Класс `DSVSettings` предоставляет настройки, необходимые для работы с файлами CSV (comma-separated value) и DSV (delimiter-separated value). Используется в `SaveDocumentSettings`, `LoadDocumentSettings`.

Описание полей класса `DSVSettings` представлено в таблице 19.

Таблица 19 – Описание полей класса `DSVSettings`

Поле	Описание
`DSVSettings.autofit`	Признак необходимости автоматического подстраивания ширины столбца под размер данных в ячейке
`DSVSettings.startBlockIndex`	Индекс блока документа сохранения
`DSVSettings.lastBlockIndex`	Индекс блока документа для окончания сохранения

Пример:

```
SaveDocumentSettings saveDocumentSettings = SaveDocumentSettings();
saveDocumentSettings.dsvSettings = DSVSettings();
saveDocumentSettings.dsvSettings.autofit = true;
saveDocumentSettings.dsvSettings.startBlockIndex = 0;
saveDocumentSettings.dsvSettings.lastBlockIndex = 10;
```

6.39 Класс Encoding

В таблице 20 приведены поддерживаемые кодировки документов. Используется в `LoadDocumentSettings`.

Таблица 20 - Кодировки документов

Наименование константы	Кодировка
`Encoding::Unknown`	Невозможно определить кодировку
`Encoding::UTF8`	UTF8
`Encoding::UTF16BE`	UTF16BE
`Encoding::UTF16LE`	UTF16LE

Наименование константы	Кодировка
`Encoding::UTF32BE`	UTF32BE
`Encoding::UTF32LE`	UTF32LE
`Encoding::Windows1250`	Windows1250
`Encoding::Windows1251`	Windows1251
`Encoding::Windows1252`	Windows1252
`Encoding::ISO8859Part5`	ISO8859Part5
`Encoding::KOI8R`	KOI8R
`Encoding::KOI8U`	KOI8U
`Encoding::CP866`	CP866

6.40 Класс ExportFormat

В таблице 21 приведены поддерживаемые форматы экспорта документов (см. `document::exportAs()`).

Таблица 21 - Форматы экспорта документов

Наименование константы	Описание
`ExportFormat::PDFA1`	Используется для работы с документами в формате Portable Document Format (PDF).

6.41 Класс Field

Класс `Field` предназначен для реализации некоторых полей, например, содержания.

6.42 Класс Fill

Класс описывает свойства заполнения фигуры: цвет заполнения, путь к изображению фона. Используется в `ShapeProperties`.

6.42.1 Метод Fill:getColor

Метод возвращает цвет заполнения `Color`.

6.42.2 Метод Fill:getUrl

Метод возвращает путь к изображению, которое используется в качестве заполнения, тип - строка.

6.42.3 Метод Fill:isNoFill

Метод возвращает `true`, если заполнения нет.

6.43 Класс FormulaType

Поддерживаемые системы адресации ячеек (стили ссылок) в табличном документе представлены в таблице 22. Используется в `document::getFormulaType()`, `document::setFormulaType()`, `DocumentSettings`.

Таблица 22 – Системы адресации ячеек в табличном документе

Наименование константы	Система адресации ячеек	Описание
`FormulaType::A1`	A1	Вариант A1 соответствует наиболее распространенной системе адресации ячеек, при которой столбцы задаются буквами, а строки – числами.
`FormulaType::R1C1`	R1C1	Вариант R1C1 соответствует альтернативной системе адресации ячеек, при которой столбцы и строки задаются числами.

6.44 Класс FractionCellFormatting

Класс содержит параметры для дробного формата ячеек таблицы. Используется в качестве аргумента метода `Cell::setFormat()`. Описание полей класса `FractionCellFormatting` представлено в таблице 23.

Таблица 23 – Описание полей класса `FractionCellFormatting`

Поле	Описание
`FractionCellFormatting.minNumeratorDigits`	Количество позиций числителя
`FractionCellFormatting.minDenominatorDigits`	Количество позиций знаменателя
`FractionCellFormatting denominatorValue`	Знаменатель

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("A2");

FractionCellFormatting cellFormat = FractionCellFormatting();
cellFormat.denominatorValue = 22;
cellFormat.minDenominatorDigits = 3;
cellFormat.minNumeratorDigits = 2;

cell.setFormat(cellFormat);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.45 Класс Frame

Класс `Frame` описывает прямоугольную область графического объекта, находящегося в текстовой позиции документа (см. Рисунок 26). Предназначен для получения и изменения свойств позиции графических объектов. Используется в текстовом документе.

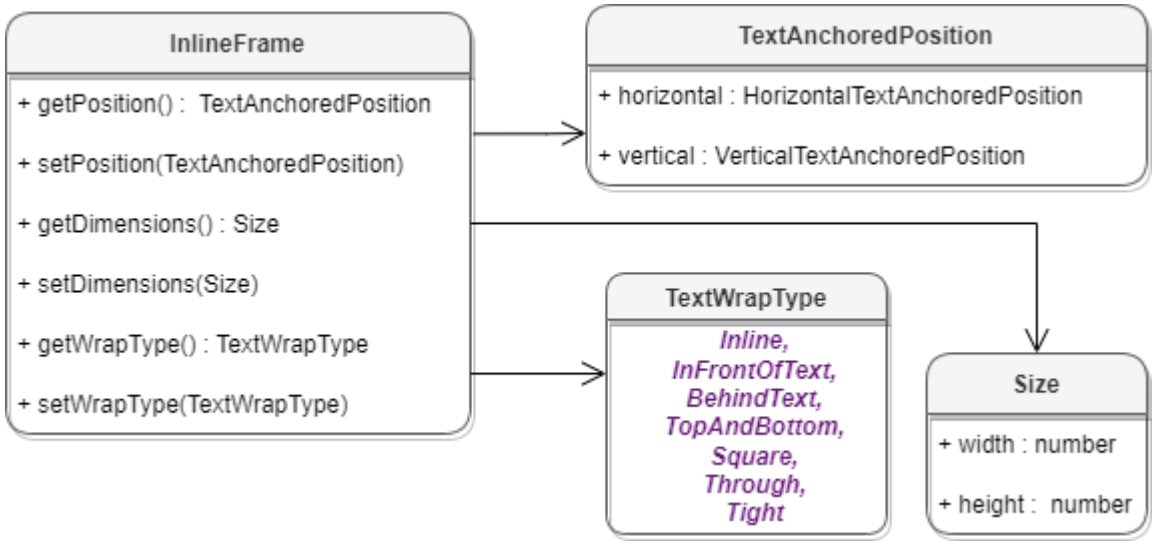


Рисунок 26 – Объектная модель класса `Frame`

Пример для текстового документа:

```
InlineObjects inlineObjects = document.getRange().getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    Frame frame = inlineObject.getFrame();
    boost::optional<TextWrapType> wrapTypeOpt = frame.getWrapType();
    if (wrapTypeOpt.has_value()) {
        std::printf("%d", wrapTypeOpt.get());
    }
    enumerator->goToNext();
}
```

6.45.1 Метод Frame:setPosition

Метод задает положение встроенного объекта, тип аргумента TextAnchoredPosition.

Внимание ! Метод может быть использован только для встроенных объектов с абсолютной позицией (у которых тип обтекания текстом не является TextWrapType::Inline). В противном случае возникнет исключение типа DocumentModificationError с ошибкой: «Set position operation is unsupported with TextWrapType::Inline».

Пример:

```
TextAnchoredPosition position = TextAnchoredPosition();

// Установка смещения по горизонтали
position.horizontal =
boost::optional
<HorizontalTextAnchoredPosition>(HorizontalRelativeTo::Character);
position.horizontal.get().offset = 10.0;

// Установка смещения по вертикали
position.vertical =
boost::optional<VerticalTextAnchoredPosition>(VerticalRelativeTo::Character);
position.vertical.get().offset = 15.0;

// Задание позиции графического объекта
frame.setPosition(position);
```

6.45.2 Метод Frame:getPosition

Метод возвращает позицию (тип `boost::optional<AnchoredPosition>`) встроенного объекта на странице.

Пример:

```
InlineObjects inlineObjects = document.getRange().getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    Frame frame = inlineObject.getFrame();
    boost::optional<AnchoredPosition> positionOpt = frame.getPosition();
    if (positionOpt.has_value()) {
        boost::optional<TextAnchoredPosition> textPositionOpt =
positionOpt.get().textPosition;
```

```
        if (textPositionOpt.has_value()) {
            std::printf("%d %d", textPositionOpt.get().horizontal,
textPositionOpt.get().vertical);
        }
    }
    enumerator->goToNext();
}
```

6.45.3 Метод Frame:setDimensions

Метод задает размер `boost::optional<Size<float>` встроенного объекта.

Пример для текстового документа:

```
InlineObjects inlineObjects = document.getRange().getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    Frame frame = inlineObject.getFrame();
    frame.setDimensions(Size<float>(50.0, 50.0));
    enumerator->goToNext();
}
```

6.45.4 Метод Frame:getDimensions

Метод возвращает задает размеры встроенного объекта, тип - `boost::optional<Size<float>>`.

Пример для текстового документа:

```
InlineObjects inlineObjects = document.getRange().getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    Frame frame = inlineObject.getFrame();
    boost::optional<Size<float>> dimensions = frame.getDimensions();
    if (dimensions.has_value()) {
        std::printf("%s", dimensions.get().toString().c_str());
    }
    enumerator->goToNext();
}
```

6.45.5 Метод Frame:getWrapType

Метод возвращает вариант обтекания текстом встроенного объекта (см. TextWrapType).

Пример для текстового документа:

```
InlineObjects inlineObjects = document.getRange().getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    Frame frame = inlineObject.getFrame();
    std::printf("%d", frame.getWrapType());
    enumerator->goToNext();
}
```

6.45.6 Метод Frame:setWrapType

Метод устанавливает вариант обтекания текстом встроенного объекта (см. TextWrapType).

Пример для текстового документа:

```
InlineObjects inlineObjects = document.getRange().getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    Frame frame = inlineObject.getFrame();
    frame.setWrapType(TextWrapType::Inline);
    enumerator->goToNext();
}
```

6.46 Класс HeaderFooter

Класс `HeaderFooter` определяет колонтитул текстового документа.

6.46.1 Метод HeaderFooter::getType

Метод предоставляет информацию о типе колонтитула (HeaderFooterType).

Пример:

```
HeaderFooter headerFooter = hfEnumerator->getCurrent();
HeaderFooterType hfType = headerFooter.getType();
std::printf("%s", hfType == HeaderFooterType::Header ? "Header" : "Footer");
```

6.46.2 Метод HeaderFooter::getBlocks

Метод предоставляет доступ к блокам (Blocks), которые содержатся в колонтитуле.

Пример:

```
Blocks blocks = headerFooter.getBlocks();
std::shared_ptr<Enumerator<Block>> blocksEnumerator = blocks.getEnumerator();
while (blocksEnumerator->isValid()) {
    Block block = blocksEnumerator->getCurrent();
    std::printf(block.getRange().extractText().c_str());
    blocksEnumerator->goToNext();
}
```

6.46.3 Метод HeaderFooter::getRange

Метод предоставляет диапазон (Range) с содержанием верхнего или нижнего колонтитулов.

Пример:

```
HeaderFooter headerFooter = hfEnumerator->getCurrent();
Range range = headerFooter.getRange();
std::printf(range.extractText().c_str());
hfEnumerator->goToNext();
```

6.47 Класс HeaderFooterType

Класс HeaderFooterType содержит типы колонтитулов – верхний колонтитул (Header) и нижний колонтитул (Footer).

```
enum class HeaderFooterType
{
    Header,
    Footer
};
```

6.48 Класс HeadersFooters

Класс `HeadersFooters` представляет коллекцию верхних и нижних колонтитулов раздела (см. Рисунок 27). Доступ к колонтитулам осуществляется посредством методов `Section::getHeaders()`, `Section::getFooters()`.

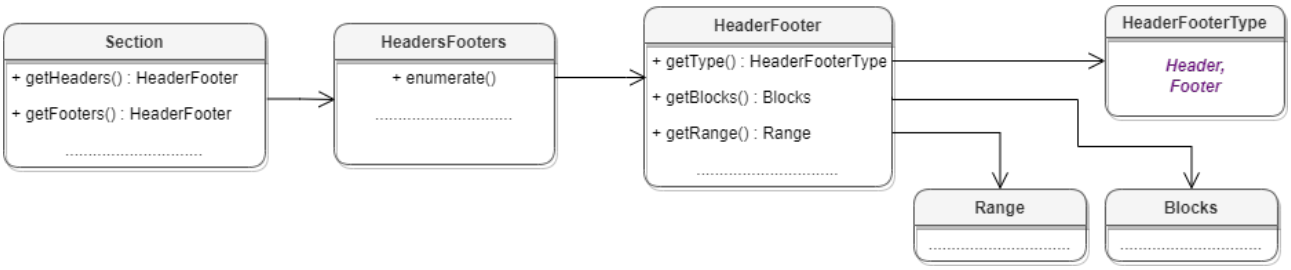


Рисунок 27 – Классы колонтитулов

6.48.1 Метод HeadersFooters::getEnumerator

Метод возвращает коллекцию колонтитулов.

Пример:

```
HeadersFooters headersFooters = section.getHeaders();
std::shared_ptr<Enumerator<HeaderFooter>> hfEnumerator =
headersFooters.getEnumerator();
while (hfEnumerator->isValid()) {
    HeaderFooter headerFooter = hfEnumerator->getCurrent();
    hfEnumerator->goToNext();
}
```

6.49 Класс HorizontalAnchorAlignment

В таблице 24 представлены типы выравнивания объекта относительно закрепленной позиции по горизонтали.

Таблица 24 – Типы выравнивания объекта относительно закрепленной позиции по горизонтали

Наименование константы	Описание
`HorizontalAnchorAlignment::Left`	По верхнему краю
`HorizontalAnchorAlignment::Right`	По нижнему краю
`HorizontalAnchorAlignment::Center`	По центру
`HorizontalAnchorAlignment::Inside`, `HorizontalAnchorAlignment::Outside`	По границам

6.50 Класс HorizontalRelativeTo

В таблице 25 представлены типы размещения объекта относительно закрепленной позиции по горизонтали.

Таблица 25 – Типы размещения объекта относительно закрепленной позиции по горизонтали

Наименование константы	Описание
`HorizontalRelativeTo::Character`	Символ
`HorizontalRelativeTo::Column`	Столбец
`HorizontalRelativeTo::ColumnLeftMargin`	Левое поле столбца
`HorizontalRelativeTo::ColumnRightMargin`	Правое поле столбца
`HorizontalRelativeTo::ColumnInsideMargin`	Внутреннее поле столбца
`HorizontalRelativeTo::ColumnOutsideMargin`	Внешнее поле столбца
`HorizontalRelativeTo::Page`	Страница
`HorizontalRelativeTo::PageContent`	Содержимое страницы
`HorizontalRelativeTo::PageLeftMargin`	Левое поле страницы
`HorizontalRelativeTo::PageRightMargin`	Правое поле страницы
`HorizontalRelativeTo::PageInsideMargin`	Внутреннее поле страницы
`HorizontalRelativeTo::PageOutsideMargin`	Внешнее поле страницы

6.51 Класс HorizontalTextAnchoredPosition

Класс `HorizontalTextAnchoredPosition` предназначен для управления относительным положением объекта со смещением или выравниванием по горизонтали.

Объекты могут быть созданы с помощью следующих констукторов:

```
HorizontalTextAnchoredPosition(HorizontalRelativeTo relativeTo);
HorizontalTextAnchoredPosition(HorizontalRelativeTo relativeTo, Unit offset);
HorizontalTextAnchoredPosition(HorizontalRelativeTo relativeTo,
HorizontalAnchorAlignment alignmentType);
```

Описание полей класса `HorizontalTextAnchoredPosition` представлено в таблице 26.

Таблица 26 – Описание полей класса `HorizontalTextAnchoredPosition`

Поле	Описание
`HorizontalTextAnchoredPosition::relativeTo`	Тип размещения объекта относительно закрепленной

Поле	Описание
	позиции по горизонтали HorizontalRelativeTo
HorizontalTextAnchoredPosition::offset	Смещение объекта
HorizontalTextAnchoredPosition::aligment	Тип выравнивания объекта относительно закрепленной позиции по горизонтали HorizontalAnchorAlignment

Примеры:

```
// Использование конструкторов
horzTextAnchoredPos =
HorizontalTextAnchoredPosition(HorizontalRelativeTo::Character);
horzTextAnchoredPos =
HorizontalTextAnchoredPosition(HorizontalRelativeTo::Character, 10.0);
horzTextAnchoredPos =
HorizontalTextAnchoredPosition(HorizontalRelativeTo::Character,
HorizontalAnchorAlignment::Inside);

// Непосредственное обращение к полям
horzTextAnchoredPos.offset = 15.0;
horzTextAnchoredPos.relativeTo = HorizontalRelativeTo::ColumnInsideMargin;
horzTextAnchoredPos.alignment = HorizontalAnchorAlignment::Inside;
```

6.51.1 Оператор ==

Метод используется для определения эквивалентности двух объектов HorizontalTextAnchoredPosition.

6.51.2 Оператор !=

Метод используется для определения неэквивалентности двух объектов HorizontalTextAnchoredPosition.

6.52 Класс LineEndingStyle

В таблице 27 приведены типы окончания линии. Используется в поле style класса LineEndingProperties.

Таблица 27 – Типы окончания линии

Наименование константы	Описание
`LineEndingStyle::Arrow`	
`LineEndingStyle::Diamond`	
`LineEndingStyle::Oval`	
`LineEndingStyle::Stealth`	
`LineEndingStyle::Triangle`	
`LineEndingStyle::None`	

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("C3");

LineProperties lineProperties = LineProperties();
lineProperties.headLineEndingProperties = LineEndingProperties();
lineProperties.headLineEndingProperties.get().style = LineEndingStyle::Arrow;

Borders borders = Borders();
borders.setTop(lineProperties);
cell.setBorders(borders);
```

6.53 Класс LineEndingProperties

Класс `LineEndingProperties` содержит варианты оформления окончаний линий. Описание полей класса `LineEndingProperties` представлено в таблице 28. Используется в полях `headLineEndingProperties` и `tailLineEndingProperties` класса `LineProperties`.

Таблица 28 – Описание полей класса `LineEndingProperties`

Поле	Тип	Описание
`LineEndingProperties.style`	`LineEndingStyle`	Стиль окончания линии
`LineEndingProperties.relativeExtent`	`Size`	Размер окончания линии относительно ее ширины

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("C3");

LineProperties lineProperties = LineProperties();
lineProperties.headLineEndingProperties = LineEndingProperties();
lineProperties.headLineEndingProperties.get().style = LineEndingStyle::Arrow;
lineProperties.headLineEndingProperties.get().relativeExtent = Size<float>();
lineProperties.headLineEndingProperties.get().relativeExtent.get().width = 2;
lineProperties.headLineEndingProperties.get().relativeExtent.get().height = 2;

lineProperties.tailLineEndingProperties = LineEndingProperties();
lineProperties.tailLineEndingProperties.get().style = LineEndingStyle::Arrow;
lineProperties.tailLineEndingProperties.get().relativeExtent = Size<float>();
lineProperties.tailLineEndingProperties.get().relativeExtent.get().width = 2;
lineProperties.tailLineEndingProperties.get().relativeExtent.get().height = 2;

lineProperties.style = LineStyle::Solid;
lineProperties.width = 1.5;
lineProperties.color = Color(ColorRGBA(55, 146, 179, 200));

Borders borders = Borders();
borders.setTop(lineProperties);
cell.setBorders(borders);
```

6.54 Класс LineProperties

Класс `LineProperties` предназначен для установки таких параметров линии, как тип, ширина, цвет (см. Рисунок 28).

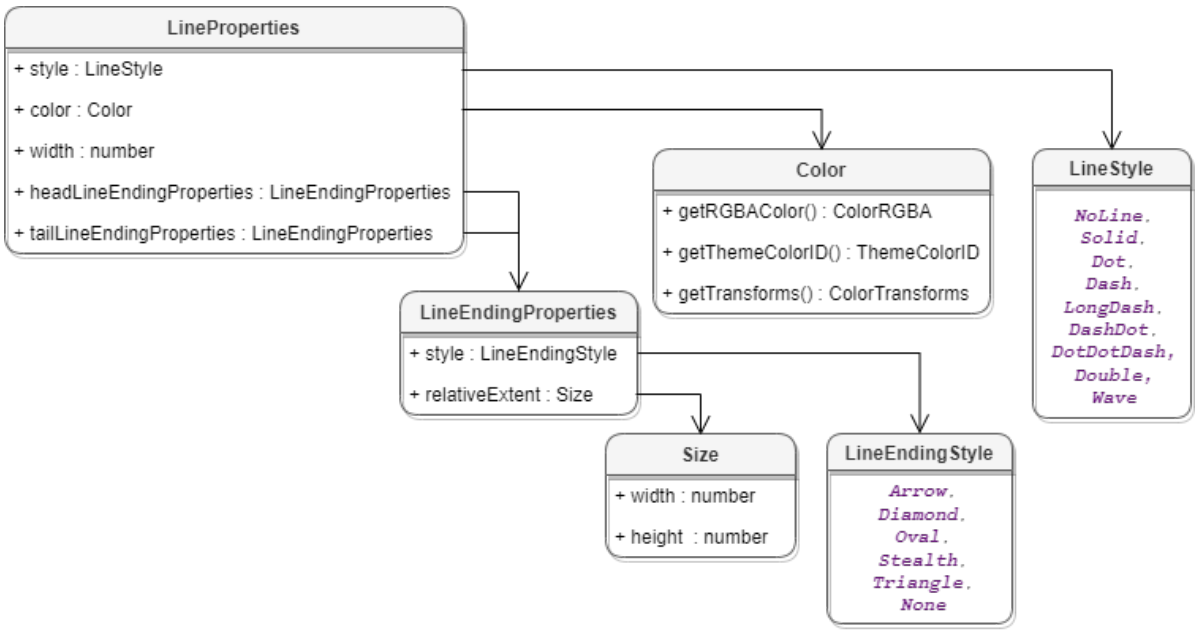


Рисунок 28 – Свойства границ ячеек

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("C3");

LineProperties lineProperties = LineProperties();
lineProperties.style = LineStyle::Solid;
lineProperties.width = 1.5;
lineProperties.color = Color(ColorRGBA(55, 146, 179, 200));

Borders borders = Borders();
borders.setTop(lineProperties);
cell.setBorders(borders);
```

6.54.1 Поле LineProperties.style

Поле предназначено для установки типа линии. Допустимые значения представлены в разделе `LineStyle`.

6.54.2 Поле LineProperties.width

Поле предназначено для установки ширины линии. Тип - числовой.

6.54.3 Поле LineProperties.color

Поле предназначено для установки цвета линии. Тип - `Color`.

6.54.4 Поле LineProperties.headLineEndingProperties

Поле предназначено для оформления начала линии `LineEndingProperties`.

6.54.5 Поле LineProperties.tailLineEndingProperties

Поле предназначено для оформления конца линии `LineEndingProperties`.

6.55 Класс LineSpacing

Класс `LineSpacing` задает межстрочный интервал абзаца. Поля класса приведены в таблице 29. Для управления значением межстрочного интервала используются значения, представленные в разделе `LineSpacingRule`.

Таблица 29 – Параметры межстрочного интервала

Поле	Описание
`LineSpacing.value`	Значение межстрочного интервала
`LineSpacing.rule`	Правило формирования межстрочного интервала `LineSpacingRule`

Пример:

```
// Конструктор
paraProps.lineSpacing = LineSpacing(5.0, LineSpacingRule::Multiple);
// Обращение к полям
paraProps.lineSpacing.value = 1;
paraProps.lineSpacing.rule = LineSpacingRule::Exact;
```

6.56 Класс LineSpacingRule

Класс `LineSpacingRule` содержит типы межстрочного интервалов.

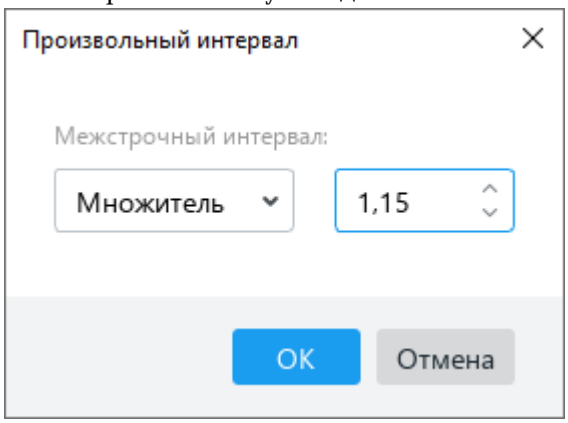
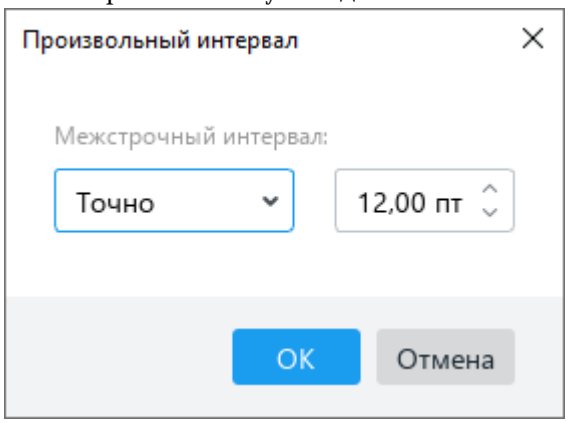
```
enum class LineSpacingRule
{
    Multiple,
    Exact,
    AtLeast
};
```

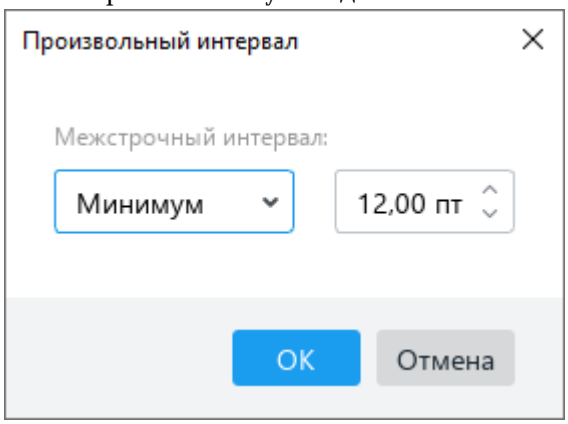
Типы межстрочных интервалов:

- `Multiple` – межстрочный интервал с использованием множителя;
- `Exact` – межстрочный интервал с использованием точного значения;
- `AtLeast` – межстрочный интервал с использованием минимального значения.

В таблице 30 представлены описания правил формирования межстрочного интервала текстового абзаца.

Таблица 30 – Виды межстрочного интервала

Наименование константы	Описание
`LineSpacingRule::Multiple`	Установка произвольного межстрочного интервала с использованием множителя. При вызове необходимо указать значение множителя, например: `pPr.lineSpacing = LineSpacing(1.15, LineSpacingRule_Multiple)` В данном примере используется значение множителя **1.15**. Действие команды аналогично ручной настройке межстрочного интервала в диалоге **«Произвольный интервал»** (см. документ «МойОфис Текст. Руководство пользователя»). 
`LineSpacingRule::Exact`	Установка произвольного межстрочного интервала с использованием точного значения. При вызове необходимо указать точное значение, например: `pPr.lineSpacing = LineSpacing(12.0, LineSpacingRule_Exact)` В данном примере используется точное значение **12.0**. Действие команды аналогично ручной настройке межстрочного интервала в диалоге **«Произвольный интервал** (см. документ «МойОфис Текст. Руководство пользователя»). 

Наименование константы	Описание
`LineSpacingRule::AtLeast`	Установка произвольного межстрочного интервала с использованием минимального значения. При вызове необходимо указать минимальное значение, например: `pPr.lineSpacing = LineSpacing(12.0, LineSpacingRule_AtLeast)` В данном примере используется точное значение **12.0**. Действие команды аналогично ручной настройке межстрочного интервала в диалоге **«Произвольный интервал»** (см. документ «МойОфис Текст. Руководство пользователя»). 

Пример:

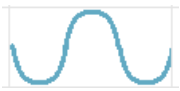
```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    ParagraphProperties paraProps = paragraphOpt.get().getParagraphProperties();
    paraProps.lineSpacing = LineSpacing(5.0, LineSpacingRule::Multiple);
}
```

6.57 Класс LineStyle

В таблице 31 приведены типы линий. Используется в поле `style` класса `LineProperties`.

Таблица 31 – Типы линий

Наименование константы	Описание
`LineStyle::NoLine`	Нет линии
`LineStyle::Solid`	
`LineStyle::Dot`	

Наименование константы	Описание
`LineStyle::Dash`	
`LineStyle::LongDash`	
`LineStyle::DashDot`	
`LineStyle::DotDotDash`	
`LineStyle::Double`	
`LineStyle::Wave`	

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("C3");

LineProperties lineProperties = LineProperties();
lineProperties.style = LineStyle::Solid;
```

6.58 Класс ListSchema

Класс `ListSchema` содержит типы схем форматирования списков, которые могут быть применены к абзацам текста. Данные константы используются в методах `Paragraph::getListSchema()`, `Paragraph::setListSchema()`.

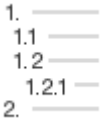
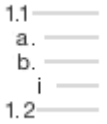
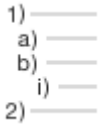
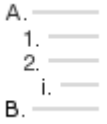
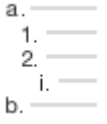
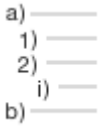
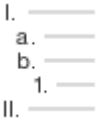
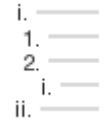
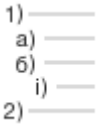
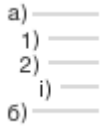
```
enum class ListSchema
{
    Unknown,
    UnknownBullet,
    UnknownNumbering,
    BulletCircleSolid,
    BulletCircleContour,
    BulletSquareSolid,
    BulletDiamondDots,
    BulletHyphen,
    BulletConcaveArrowSolid,
    BulletCheckmark,
    EnumeratorDecimalDot,
```

```
    EnumeratorDecimalDotMultiLevel,
    EnumeratorDecimalBracket,
    EnumeratorLatinUppercaseDot,
    EnumeratorLatinLowercaseDot,
    EnumeratorLatinLowercaseBracket,
    EnumeratorRomanUppercaseDot,
    EnumeratorRomanLowercaseDot,
    EnumeratorDecimalRussianBracket,
    EnumeratorRussianLowercaseBracket
};
```

Описания схем форматирования списков представлены в таблице 32.

Таблица 32 – Описания схем списков

Тип схемы списка	Описание типа схемы списка	Изображение
`Unknown`	Неизвестно	
`UnknownBullet`	Список без маркера	Соответствует варианту «нет»
`UnknownNumbering`	Нумерация без номера	Соответствует варианту «нет»
`BulletCircleSolid`	Список с маркерами в виде круга	
`BulletCircleContour`	Список с маркерами в виде окружности	
`BulletSquareSolid`	Список с маркерами в виде квадрата	
`BulletDiamondDots`	Список с маркерами в виде четырех ромбов	
`BulletHyphen`	Список с маркерами в виде дефиса	
`BulletConcaveArrowSolid`	Список с маркерами в виде вогнутой стрелки	

Тип схемы списка	Описание типа схемы списка	Изображение
`BulletCheckmark`	Список с маркерами в виде галочки	
`EnumeratorDecimalDot`	Десятичная нумерация с точкой	
`EnumeratorDecimalDotMultiLevel`	Многоуровневая десятичная нумерация с точкой	
`EnumeratorDecimalBracket`	Десятичная нумерация со скобкой	
`EnumeratorLatinUppercaseDot`	Нумерация латинскими прописными буквами с точкой	
`EnumeratorLatinLowercaseDot`	Нумерация латинскими строчными буквами с точкой	
`EnumeratorLatinLowercaseBracket`	Нумерация латинскими строчными буквами со скобкой	
`EnumeratorRomanUppercaseDot`	Нумерация римскими прописными цифрами с точкой	
`EnumeratorRomanLowercaseDot`	Нумерация римскими строчными цифрами с точкой	
`EnumeratorDecimalRussianBracket`	Десятичная нумерация через запятую со скобкой	
`EnumeratorRussianLowercaseBracket`	Нумерация с русскими строчными буквами со скобкой	

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    Paragraph paragraph = paragraphOpt.get();
    paragraph.setListSchema(ListSchema::BulletCircleContour);
}
```

6.59 Класс LoadDocumentSettings

Класс `LoadDocumentSettings` предоставляет дополнительные настройки, необходимые для загрузки документов из файла (см. `application::loadDocument()`).

Описание полей класса `LoadDocumentSettings` представлено в таблице 33.

Таблица 33 – Описание полей класса `LoadDocumentSettings`

Поле	Описание
`LoadDocumentSettings.commonDocumentSettings`	Экземпляр таблицы, общие настройки документа `DocumentSettings`.
`LoadDocumentSettings.encoding`	Кодировка документа `Encoding`.
`LoadDocumentSettings.dsvSettings`	Экземпляр класса `DSVSettings`, настройки, необходимые для работы с файлами CSV и DSV .
`LoadDocumentSettings.documentPassword`	Пароль для защиты электронного документа от несанкционированного доступа. Механизм парольной защиты поддерживается только для семейства ОС Microsoft Windows.

6.60 Класс Image

Класс `Image` представляет собой изображение, находящееся в текстовом или табличном документе.

6.60.1 Метод Image:getFrame

Метод аналогичен методу `MediaObject::getFrame()`, он возвращает свойства позиции изображения `Frame`.

Пример:

```
Image image = enumerator->getCurrent();
Frame frame = image.getFrame();
boost::optional < CO::API::Size<float>> dimensionsOpt = frame.getDimensions();
if (dimensionsOpt.has_value()) {
    Size<float> size = dimensionsOpt.get();
    std::printf("%f", size.width);
}
```

6.61 Класс Images

Класс `Images` используется для доступа к коллекции изображений. Объект может быть получен посредством вызова метода `Range::getImages()`.

6.61.1 Метод Images:enumerate

Метод позволяет перечислить коллекцию изображений.

Пример:

```
CO::API::Document::Images images = document.getRange().getImages();
std::shared_ptr<Enumerator<Image>> enumerator = images.getEnumerator();
while (enumerator->isValid()) {
    Image image = enumerator->getCurrent();
    Frame frame = image.getFrame();
    boost::optional < CO::API::Size<float>> dimensionsOpt =
frame.getDimensions();
    if (dimensionsOpt.has_value()) {
        Size<float> size = dimensionsOpt.get();
        std::printf("%f", size.width);
    }
    enumerator->goToNext();
}
```

6.62 Класс InlineObject

Класс `InlineObject` представляет собой графический объект документа.

6.62.1 Метод InlineObject:toImage

Метод возвращает изображение `Image`, связанное со встроенным объектом. Если объект не является изображением, метод возвращает `nil`.

Пример:

```
Range range = document.getRange();
InlineObjects inlineObjects = range.getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    boost::optional<Image> imageOpt = inlineObject.toImage();
    if (imageOpt.has_value()) {
        // Image
        Image image = imageOpt.get();
    } else {
        // Not an image
    }
    enumerator->goToNext();
}
```

6.62.2 Метод InlineObject:getFrame

Метод возвращает свойства позиции встроенного объекта `Frame`.

Пример:

```
Range range = document.getRange();
InlineObjects inlineObjects = range.getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    Frame frame = inlineObject.getFrame();
    boost::optional<Size<float>> dimensionsOpt = frame.getDimensions();
    ..........................
    enumerator->goToNext();
}
```

6.63 Класс InlineObjects

Класс `InlineObjects` предназначен для доступа к коллекции графических объектов. Объект может быть получен вызовом метода `Range::getInlineObjects()` (см. Рисунок 29).

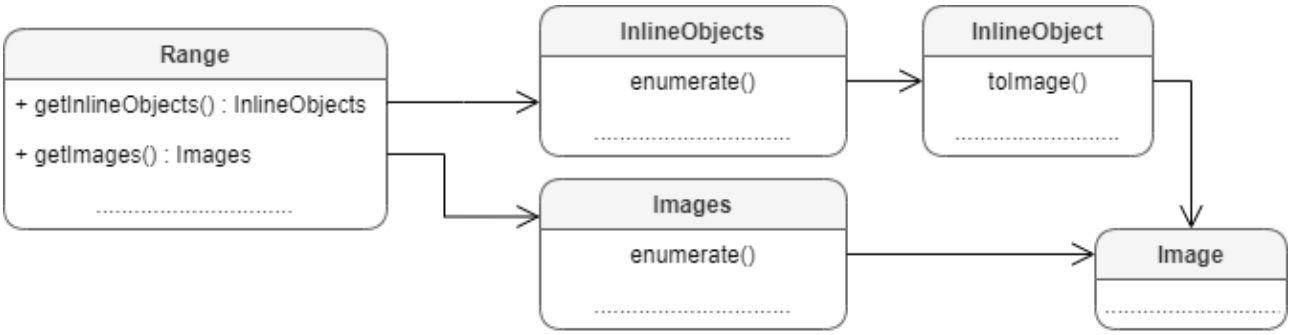


Рисунок 29 – Графические объекты

6.63.1 Метод InlineObjects::getEnumerator

Метод позволяет перечислить коллекцию графических объектов.

Пример:

```
Range range = document.getRange();
InlineObjects inlineObjects = range.getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    boost::optional<Size<float>> dimensions =
inlineObject.getFrame().getDimensions();
    if (dimensions.has_value()) {
        std::printf("%d", dimensions.get().height);
    }
}
```

6.64 Класс Insets

Класс `Insets` предназначен для задания полей, например, страницы. Поля класса `Insets` представлены в таблице 34. Используется в поле `margins` класса `PageProperties`.

Таблица 34 – Описание полей класса `Insets`

Поле	Тип	Описание
`Insets.left`	`boost::optional<Unit>`	Левая граница поля
`Insets.top`	`boost::optional<Unit>`	Верхняя граница поля
`Insets.right`	`boost::optional<Unit>`	Правая граница поля
`Insets.bottom`	`boost::optional<Unit>`	Нижняя граница поля

Пример:

```
PageProperties pageProperties = PageProperties();
Insets insets = Insets();
insets.left = 0;
insets.top = 0;
insets.right = 100;
insets.bottom = 100;
pageProperties.margins = insets;
document.setPageProperties(pageProperties);
```

6.65 Класс LocaleInfo

Класс `LocaleInfo` предоставляет информацию о локализации. Используется в поле `localeInfo` класса `DocumentSettings`.

Описание полей `LocaleInfo` представлено в таблице 35.

Таблица 35 – Описание полей класса `LocaleInfo`

Поле	Описание
`LocaleInfo.localeName`	Название локализации, представлено в формате **<language> <REGION>**, где языковой код соответствует стандарту ISO-639, а код региона стандарту ISO-3166.
`LocaleInfo.decimalSeparator`	Десятичный разделитель, отделяет целые и дробные части чисел.
`LocaleInfo.thousandSeparator`	Символ, разделяющий группы цифр в числовых значениях.
`LocaleInfo.listSeparator`	Символ, отделяющий элементы в списке.
`LocaleInfo.currencySymbol`	Символ валюты, используемой в текущей стране или регионе.
`LocaleInfo.currencyFormat`	Расположение знака валюты в текущем регионе, тип: `CurrencySignPlacement`.
`LocaleInfo.shortDatePattern`	Заданный «короткий» формат отображения даты в текущем регионе (например, 'm/d/yy' для en_US).
`LocaleInfo.longDatePattern`	Заданный «длинный» формат отображения даты в текущем регионе (например, 'dddd, mmmm d, yyyy' for en_US).
`LocaleInfo.timePattern`	Заданный формат отображения времени в текущем регионе (например, 'h:mm AM/PM' для en_US).

6.66 Класс Message

Класс `Message` предназначен для формирования событий лога.

6.66.1 Класс Message::Severity

Класс `Message::Severity` (Таблица 36) описывает уровни сообщений лога (информация, предупреждение, ошибка).

Таблица 36 – Описание уровней лога `Message::Severity`

Поле	Описание
`Message::Severity::Info`	Информация
`Message::Severity::Warning`	Предупреждение
`Message::Severity::Error`	Ошибка

6.66.2 Метод Message::getSeverity

Метод возвращает уровень лога `Message::Severity`.

6.66.3 Метод Message::getText

Метод возвращает текст сообщения.

6.66.4 Метод Message::makeInfo

Метод создает сообщение типа `Message::Severity::Info` с заданным текстом.

6.66.5 Метод Message::makeWarning

Метод создает сообщение типа `Message::Severity::Warning` с заданным текстом.

6.66.6 Метод Message::makeError

Метод создает сообщение типа `Message::Severity::Error` с заданным текстом.

6.67 Класс Messenger

6.67.1 Метод Messenger:subscribe

Метод служит для подписки на события лога.

Пример:

```
Messenger::MessageHandlerFunction handler;
std::shared_ptr<Messenger> messenger = application.getMessenger();
std::shared_ptr<Connection> connection = messenger->subscribe(handler);
```

6.67.2 Метод Messenger:notify

Метод используется для создания события лога

Пример:

```
std::shared_ptr<Messenger> messenger = application.getMessenger();
messenger->notify(Message::makeWarning("Warning"));
```

6.68 Класс NamedExpressions

Класс для представления списка именованных диапазонов. Объект `NamedExpresions` может быть получен с помощью методов `Document::getNamedExpressions()`, `Table::getNamedExpressions()`.

6.68.1 Метод NamedExpressions::get

Возвращает именованный диапазон `NamedExpression` по имени `name`, если он существует.

Пример:

```
NamedExpressions namedExpressions = firstSheet.getNamedExpressions();
boost::optional<NamedExpression> namedExpressionOpt =
namedExpressions.get("Продажи");
if (namedExpressionOpt.has_value()) {
    std::printf("%s", namedExpressionOpt.get().getName().c_str());
}
```

6.68.2 Метод NamedExpressions::enumerate

Позволяет получить доступ ко всему списку именованных диапазонов.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
NamedExpressions namedExpressions = firstSheet.getNamedExpressions();
std::shared_ptr<Enumerator<NamedExpression>> enumerator =
namedExpressions.getEnumerator();
while (enumerator->isValid()) {
    NamedExpression namedExpression = enumerator->getCurrent();
    std::printf("%s", namedExpression.getName().c_str());
    enumerator->goToNext();
}
```

6.68.3 Метод NamedExpression::addExpression

Добавляет новый именованный диапазон, возвращает результат операции NamedExpressionsValidationResult.

Пример:

```
NamedExpressions namedExpressions = firstSheet.getNamedExpressions();
std::string expressionName = "Покупки";
std::string expressionValue = "=Формула покупки!$E$6:$E$14";
NamedExpressionsValidationResult validationResult =
namedExpressions.addExpression(expressionName, expressionValue);
std::printf("%d", validationResult);
```

6.68.4 Метод NamedExpressions::removeExpression

Удаляет именованный диапазон по заданному имени, возвращает результат операции NamedExpressionsValidationResult.

Пример:

```
std::string expressionName = "Покупки";
boost::optional<NamedExpression> namedExpressionOpt =
namedExpressions.get(expressionName);
if (namedExpressionOpt.has_value()) {
    NamedExpressionsValidationResult validationResult =
namedExpressions.removeExpression(expressionName);
    std::printf("%d", validationResult);
}
```

6.69 Класс NamedExpression

Класс описывает структуру именованного диапазона.

Пример:

```
std::shared_ptr<Enumerator<NamedExpression>> enumerator =
namedExpressions.getEnumerator();
while (enumerator->isValid()) {
    NamedExpression namedExpression = enumerator->getCurrent();
    std::printf("%s", namedExpression.getName().c_str());
    std::printf("%s", namedExpression.getExpression().c_str());
    boost::optional <CellRange> cellRangeOpt = namedExpression.getCellRange();
    if (cellRangeOpt.has_value()) {
        CellRange cellRange = cellRangeOpt.get();
```

```
        std::printf("%d-%d", cellRange.getBeginColumn(),
cellRange.getLastColumn());
    }
    enumerator->goToNext();
}
```

6.69.1 Метод NamedExpression::getName

Возвращает имя именованного диапазона. Пример см. в NamedExpression.

6.69.2 Метод NamedExpression::getExpression

Возвращает текст именованного диапазона (формулы). Пример см. в NamedExpression.

6.69.3 Метод NamedExpression::getCellRange

Возвращает именованный диапазон ячеек CellRange. Пример см. в NamedExpression.

6.70 Класс NamedExpressionsValidationResult

Класс NamedExpressionsValidationResult описывает результат операций NamedExpressions::addExpression(), NamedExpressions::removeExpression().

```
enum class NamedExpressionsValidationResult
{
    Success,
    WrongName,
    IsUsedInFormula,
};
```

Класс содержит следующие поля:

- Success – операция выполнена успешно;
- WrongName – неправильный формат имени;
- IsUsedInFormula – имя уже используется в формуле.

6.71 Класс NumberCellFormatting

Класс содержит параметры для числового формата ячеек таблицы, используется в качестве аргумента метода Cell::setFormat(). Описание полей класса NumberCellFormatting представлено в таблице 37.

Таблица 37 – Описание полей класса NumberCellFormatting

Поле	Описание
NumberCellFormatting.decimalPlaces	Количество десятичных позиций
NumberCellFormatting.useThousandsSeparator	Использовать разделитель для тысячных
NumberCellFormatting.useRedForNegative	Использовать красный цвет для отрицательных значений
NumberCellFormatting.useBracketsForNagative	Использовать скобки для отрицательных значений
NumberCellFormatting.hideSign	Скрывать знак «минус» для отрицательных значений

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("A2");

NumberCellFormatting cellFormat = NumberCellFormatting();
cellFormat.decimalPlaces = 2;
cellFormat.useThousandsSeparator = true;
cellFormat.useRedForNegative = true;
cellFormat.useBracketsForNegative = true;
cellFormat.hideSign = false;

cell.setFormat(cellFormat);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.72 Класс PageFieldOrder

Класс PageFieldOrder описывает вид отображения полей из области фильтров. Является полем класса PivotTableLayoutSettings. Описание полей класса представлено в таблице 38.

Таблица 38 – Описание полей класса PageFieldOrder

Поле	Описание
PageFieldOrder::DownThenOver	Вниз, затем поперек
PageFieldOrder::OverThenDown	Поперек, затем вниз

6.73 Класс PageNumbers

Класс `PageNumbers` используется в качестве поля `pageNumbers` класса `TextExportSettings` и представляет собой коллекцию страниц для экспорта.

Позволяет установить следующие типы страниц для экспорта:

- нечетные, четные страницы, тип `PageParity`;
- список конкретных номеров страниц, тип `std::vector<size_t>`;
- диапазон страниц с указанием начальной и конечной страницы.

Примеры:

```
// четные страницы
PageNumbers pageNumbers = PageNumbers(PageParity::Even);
```

```
// конкретные номера страниц
auto pages = std::vector<size_t>(3);
pages[0] = 1;
pages[1] = 13;
pages[2] = 25;
pageNumbers = PageNumbers(pages);
```

```
// диапазон страниц
pageNumbers = PageNumbers(1, 20);
```

6.73.1 Метод PageNumbers::contains

Метод служит для проверки вхождения заданного номера страницы в коллекцию номеров страниц `PageNumbers`.

Пример:

```
PageNumbers pageNumbers = PageNumbers(1, 20);
std::printf("%d", pageNumbers.contains(2));
```

6.73.2 Метод PageNumbers::getLast

Метод `PageNumbers::getLast` возвращает последний номер страницы.

Пример:

```
PageNumbers pageNumbers = PageNumbers(1, 20);
std::printf("%d", pageNumbers.getLast());
```

6.74 Класс PageOrientation

Тип `PageOrientation` определяет варианты ориентации страницы документа: Альбомная (Landscape) или Книжная (Portrait). Может быть использована для получения / установки ориентации страниц для секции или документа.

```
enum class PageOrientation : std::uint8_t
{
    Landscape,
    Portrait
};
```

Примеры:

```
boost::optional<Block> blockOpt = document.getBlocks().getBlock(0);
if (blockOpt.has_value()) {
    Section section = blockOpt.get().getSection();
    section.setPageOrientation(PageOrientation::Landscape);
    boost::optional<PageOrientation> pageOrientationOpt =
section.getPageOrientation();
    if (pageOrientationOpt.has_value()) {
        PageOrientation pageOrientation = pageOrientationOpt.get();
        std::printf("%s", pageOrientation == PageOrientation::Portrait ?
"Portrait" : "Landscape");
    }
}
```

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    Section section = enumerator->getCurrent();
    section.setPageOrientation(PageOrientation::Landscape);
    boost::optional<PageOrientation> pageOrientationOpt =
section.getPageOrientation();
    if (pageOrientationOpt.has_value()) {
        PageOrientation pageOrientation = pageOrientationOpt.get();
        std::printf("%s", pageOrientation == PageOrientation::Portrait ?
"Portrait" : "Landscape");
    }
}
```

6.75 Класс PageProperties

Класс `PageProperties` предоставляет такие свойства страницы как высота, ширина, размеры полей. Описание полей приведено в таблице 39. Используется в `Document::setPageProperties()`, `Section::getPageProperties()`, `Section::setPageProperties()`.

Таблица 39 – Описание полей класса `PageProperties`

Поле	Описание
`PageProperties.height`	Высота страницы
`PageProperties.width`	Ширина страницы
`PageProperties.margins`	Поля страницы, тип - `Insets`

Примеры:

```
PageProperties pageProperties = section.getPageProperties();
pageProperties.height = 100;
pageProperties.width = 200;
section.setPageProperties(pageProperties);
```

```
PageProperties pageProperties = PageProperties();
pageProperties.height = 100;
pageProperties.width = 200;
document.setPageProperties(pageProperties);
```

```
PageProperties pageProperties = PageProperties(100, 200);
document.setPageProperties(pageProperties);
```

6.75.1 Оператор ==

Оператор сравнения **==** используется для определения эквивалентности значений двух структур `PageProperies`.

```
bool operator == (const PageProperties& other) const;
```

6.75.2 Оператор !=

Оператор сравнения **!=** используется для определения неэквивалентности значений двух структур `PageProperies`.

```
bool operator != (const PageProperties& other) const;
```

6.76 Класс PageParity

Варианты выбора страниц для экспорта и печати представлены в таблице 40.

Используется в PageNumbers, PrintSettings.

Таблица 40 – Варианты выбора страниц для экспорта и печати

Наименование константы	Описание
PageParity::Odd	Только нечетные страницы
PageParity::Even	Только четные страницы
PageParity::All	Все страницы

6.77 Класс Paragraph

Класс Paragraph предоставляет доступ к свойствам абзаца.

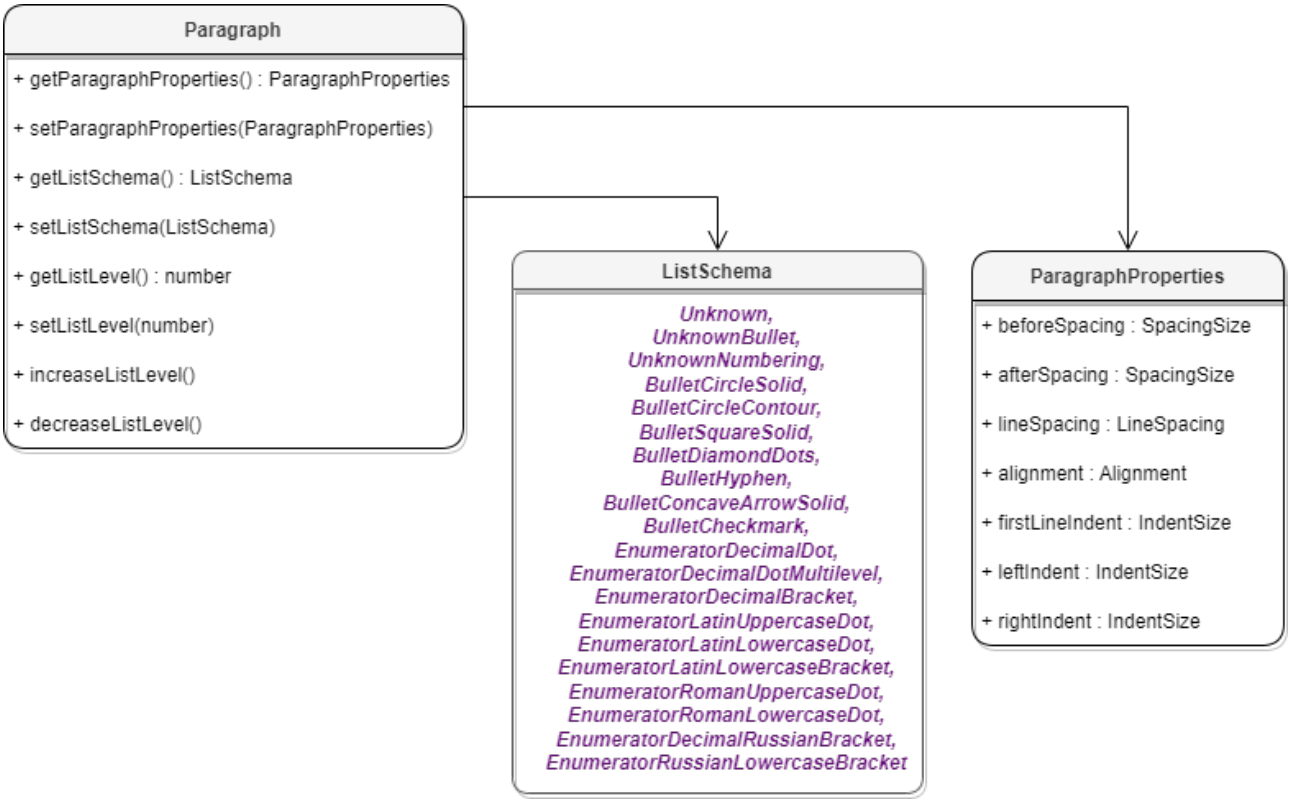


Рисунок 30 – Объектная модель классов для работы со свойствами параграфа

6.77.1 Метод Paragraph::getParagraphProperties

Метод предоставляет доступ к классу, определяющему такие свойства абзаца ParagraphProperties, как выравнивание текста, межстрочные интервалы, отступы и т. д.

Пример для текстового документа:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    Paragraph paragraph = paragraphOpt.get();
```

```
    ParagraphProperties paragraphProperties =
paragraph.getParagraphProperties();
    std::printf("%d", paragraphProperties.alignment);
}
```

Пример для табличного документа:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("B2");
Range range = cell.getRange();
Paragraphs paragraphs = range.getParagraphs();
std::shared_ptr<Enumerator<Paragraph>> enumerator = paragraphs.getEnumerator();
while (enumerator->isValid()) {
    Paragraph paragraph = enumerator->getCurrent();
    ParagraphProperties paragraphProperties =
paragraph.getParagraphProperties();
    std::printf("%d", paragraphProperties.alignment);
    enumerator->goToNext();
}
```

6.77.2 Метод Paragraph::setParagraphProperties

Метод предназначен для обновления свойств форматирования абзаца ParagraphProperties.

Пример для текстового документа:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    Paragraph paragraph = paragraphOpt.get();
    ParagraphProperties paragraphProperties =
paragraph.getParagraphProperties();
    paragraphProperties.alignment = Alignment::Center;
    paragraph.setParagraphProperties(paragraphProperties);
}
```

Пример для табличного документа:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("B2");
Range range = cell.getRange();
```

```
Paragraphs paragraphs = range.getParagraphs();
std::shared_ptr<Enumerator<Paragraph>> enumerator = paragraphs.getEnumerator();
while (enumerator->isValid()) {
    Paragraph paragraph = enumerator->getCurrent();
    ParagraphProperties paragraphProperties =
paragraph.getParagraphProperties();
    paragraphProperties.alignment = Alignment::Center;
    paragraph.setParagraphProperties(paragraphProperties);
    enumerator->goToNext();
}
```

6.77.3 Метод Paragraph::getListSchema

Метод возвращает схему форматирования абзаца `ListSchema`, если схема нумерации установлена для абзаца. Данный метод используется только в текстовом документе.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    Paragraph paragraph = paragraphOpt.get();
    boost::optional<ListSchema> listSchemaOpt = paragraph.getListSchema();
    if (listSchemaOpt.has_value()) {
        ListSchema listSchema = listSchemaOpt.get();
    }
}
```

6.77.4 Метод Paragraph::setListSchema

Метод позволяет установить тип маркированного или нумерованного списка `ListSchema`. Данный метод используется только в текстовом документе.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    Paragraph paragraph = paragraphOpt.get();
    paragraph.setListSchema(ListSchema::BulletCircleContour);
}
```

6.77.5 Метод Paragraph::getListLevel

Метод позволяет получить глубину вложенности элемента списка. Данный метод используется только в текстовом документе.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    Paragraph paragraph = paragraphOpt.get();
    std::printf("%d", paragraph.getListLevel());
}
```

6.77.6 Метод Paragraph::setListLevel

Метод позволяет установить глубину вложенности элемента списка.

Значение может быть не определено (`boost::none`), если схема нумерации не установлена для абзаца. В этом случае будет установлено минимальное значение. Данный метод используется только в текстовом документе.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    Paragraph paragraph = paragraphOpt.get();
    paragraph.setListLevel(boost::none);
}
```

6.77.7 Метод Paragraph::increaseListLevel

Метод позволяет увеличить на единицу глубину вложенности элемента списка. В случае, если максимальный уровень уже установлен, увеличения не происходит. Данный метод используется только в текстовом документе.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    paragraphOpt.get().increaseListLevel();
}
```

6.77.8 Метод Paragraph::decreaseListLevel

Метод позволяет уменьшить на единицу глубину вложенности элемента списка. В случае, если минимальный уровень уже установлен, уменьшения не происходит. Данный метод используется только в текстовом документе.

Пример:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    paragraphOpt.get().decreaseListLevel();
}
```

6.78 Класс Paragraphs

Класс `Paragraphs` предоставляет доступ к коллекции абзацев типа `Paragraph` (см. Рисунок 31). Коллекция абзацев может быть получена из объекта `Range` посредством использования метода `Range::getParagraphs()`.

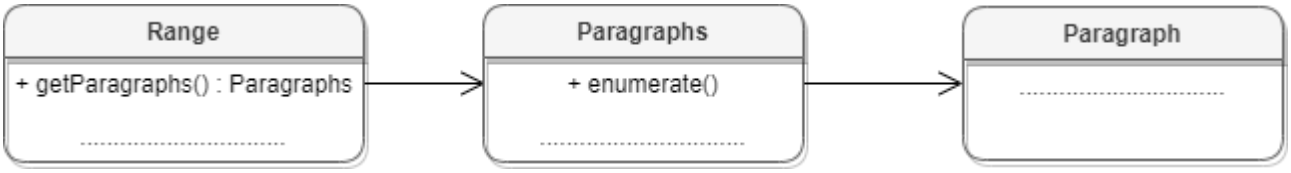


Рисунок 31 – Объектная модель для работы со списком абзацев

Пример для текстового документа:

```
Range range = document.getRange();
Paragraphs paragraphs = range.getParagraphs();
```

Пример для табличного документа:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("B2");
Range range = cell.getRange();
Paragraphs paragraphs = range.getParagraphs();
```

6.78.1 Метод Paragraphs::setListSchema

Метод устанавливает тип маркированного или нумерованного списка `ListSchema`. Данный метод используется только в текстовом документе.

Пример:

```
Range range = document.getRange();
Paragraphs paragraphs = range.getParagraphs();
paragraphs.setListSchema(ListSchema::BulletCheckmark);
```

6.78.2 Метод Paragraphs::setListLevel

Метод устанавливает глубину вложенности элемента списка. Данный метод используется только в текстовом документе.

Пример:

```
Range range = document.getRange();
Paragraphs paragraphs = range.getParagraphs();
paragraphs.setListLevel(1);
```

6.78.3 Метод Paragraphs::increaseListLevel

Метод увеличивает уровень списка на единицу. В случае, если максимальный уровень уже установлен, увеличения не происходит. Данный метод используется только в текстовом документе.

Пример:

```
Range range = document.getRange();
Paragraphs paragraphs = range.getParagraphs();
paragraphs.increaseListLevel();
```

6.78.4 Метод Paragraphs::decreaseListLevel

Метод уменьшает уровень списка на единицу. В случае, если минимальный уровень уже установлен, уменьшения не происходит. Данный метод используется только в текстовом документе.

Пример:

```
Range range = document.getRange();
Paragraphs paragraphs = range.getParagraphs();
paragraphs.decreaseListLevel();
```

6.78.5 Метод Paragraphs::getEnumerator

Метод позволяет перечислить коллекцию абзацев.

Пример:

```
std::shared_ptr<Enumerator<Paragraph>> enumerator = paragraphs.getEnumerator();
while (enumerator->isValid()) {
    Paragraph paragraph = enumerator->getCurrent();
    ParagraphProperties paragraphProperties =
paragraph.getParagraphProperties();
    std::printf("%d", paragraphProperties.alignment);
    enumerator->goToNext();
}
```

6.79 Класс ParagraphProperties

Класс `ParagraphProperties` предназначен для управления свойствами форматирования (см. Рисунок 32). Класс `ParagraphProperties` используется в методах `Paragraph::getParagraphProperties` и `Paragraph::setParagraphProperties`.

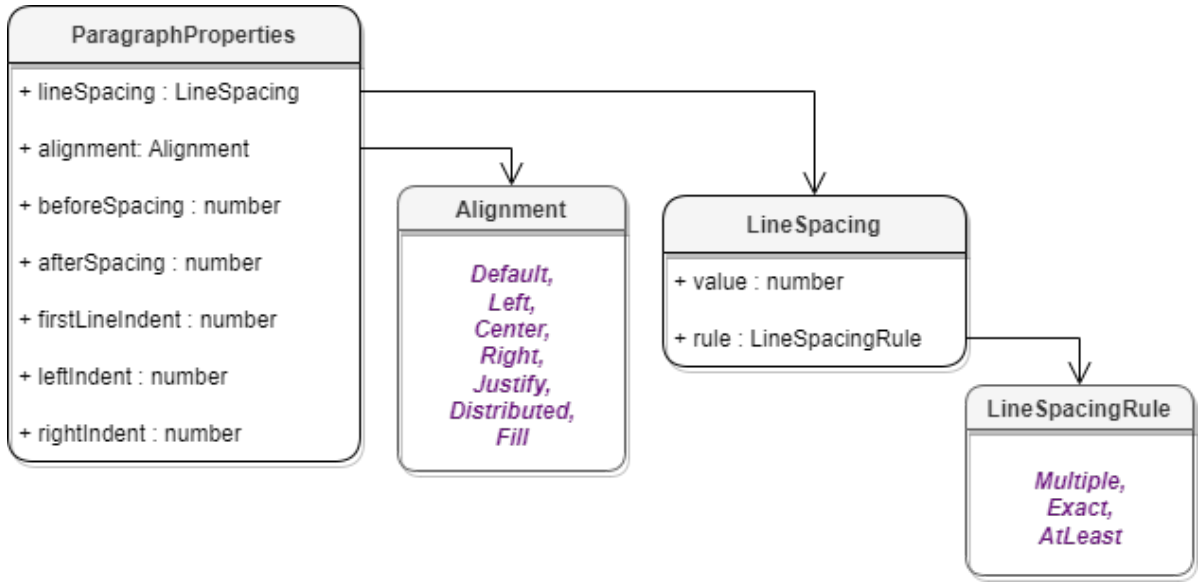
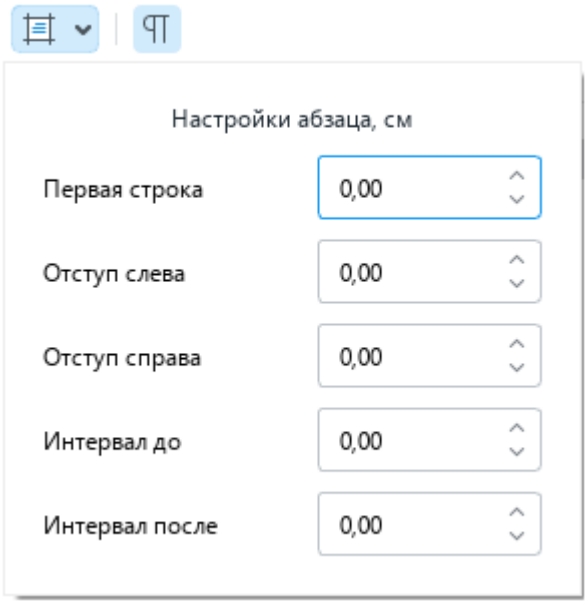


Рисунок 32 – Объектная модель классов для работы со свойствами параграфа

Описание полей класса `ParagraphProperties` представлено в таблице 41.

Таблица 41 – Описание полей класса `ParagraphProperties`

Поле	Описание
`ParagraphProperties.beforeSpacing`	Установка величины расстояния до абзаца.

Поле	Описание
	При работе с пользовательским интерфейсом приложения соответствует значению, указанному в диалоговом окне **Настройки абзаца,** (см. рисунок выше), в поле **Интервал до** (подробнее см. в документе «МойОфис Текст. Руководство пользователя»). 
`ParagraphProperties.afterSpacing`	Установка величины расстояния после абзаца. При работе с пользовательским интерфейсом приложения соответствует значению, указанному в диалоговом окне **Настройки абзаца**, в поле **Интервал после** (подробнее см. в документе «МойОфис Текст. Руководство пользователя»).
`ParagraphProperties.lineSpacing`	Расстояние между строк одного абзаца (межстрочный интервал), `LineSpacing`.
`ParagraphProperties.alignment`	Выравнивание текстового фрагмента по горизонтали. Список допустимых значений находится в разделе `Alignment`.
`ParagraphProperties.firstLineIndent`	Расстояние от левого поля документа до первой строки в абзаце с учетом отступа слева. При работе с пользовательским интерфейсом приложения соответствует значению, указанному в диалоговом окне **Настройки абзаца**, (см. рисунок выше), в поле **Первая строка** (подробнее см. в документе «МойОфис Текст. Руководство пользователя»).
`ParagraphProperties.leftIndent`	Расстояние от левого поля документа до абзаца (отступ слева). При работе с пользовательским интерфейсом приложения соответствует значению, указанному

Поле	Описание
	в диалоговом окне **Настройки абзаца**, (см. рисунок выше), в поле **Отступ слева** (подробнее см. в документе «МойОфис Текст. Руководство пользователя»).
`ParagraphProperties.rightIndent`	Расстояние от правого поля документа до абзаца. При работе с пользовательским интерфейсом приложения соответствует значению, указанному в диалоговом окне **Настройки абзаца**, (см. рисунок выше), в поле **Отступ справа** (подробнее см. в документе «МойОфис Текст. Руководство пользователя»).

Пример для текстового документа:

```
Blocks blocks = document.getBlocks();
boost::optional<Paragraph> paragraphOpt = blocks.getParagraph(0);
if (paragraphOpt.has_value()) {
    ParagraphProperties paraProps = paragraphOpt.get().getParagraphProperties();
    paraProps.afterSpacing = 28.3;    // соответствует 1 см
    paraProps.beforeSpacing = 28.3;   // соответствует 1 см
    paraProps.firstLineIndent = 28.3; // соответствует 1 см
    paraProps.leftIndent = 28.3;      // соответствует 1см
    paraProps.rightIndent = 28.3;     // соответствует 1см
    paraProps.alignment = Alignment::Center;
    paraProps.lineSpacing = LineSpacing(5.0, LineSpacingRule::Multiple);
}
```

Пример для табличного документа:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("B2");
Range range = cell.getRange();
Paragraphs paragraphs = range.getParagraphs();
std::shared_ptr<Enumerator<Paragraph>> enumerator = paragraphs.getEnumerator();
while (enumerator->isValid()) {
    Paragraph paragraph = enumerator->getCurrent();
    ParagraphProperties paragraphProperties =
paragraph.getParagraphProperties();
    std::printf("%d", paragraphProperties.alignment);
    enumerator->goToNext();
}
```

6.80 Класс PercentageCellFormatting

Содержит параметр для процентного формата ячеек таблицы, используется в качестве аргумента метода `Cell::setFormat()`. Описание полей класса `PercentageCellFormatting` представлено в таблице 42.

Таблица 42 – Описание полей класса `PercentageCellFormatting`

Поле	Описание
`PercentageCellFormatting.decimalPlaces`	Количество десятичных позиций

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("A2");

PercentageCellFormatting cellFormat = PercentageCellFormatting();
cellFormat.decimalPlaces = 2;

cell.setFormat(cellFormat);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.81 Класс PivotTable

Класс для представления сводной таблицы. Может быть получен из ячейки `Cell::getPivotTable()`, либо при создании новой сводной таблицы `PivotTablesManager::create()`.

6.81.1 Метод PivotTable::remove

Метод удаляет сводную таблицу.

Пример:

```
Table sheet = document.getBlocks().getTable(0).get();
Cell cell = sheet.getCell("A1");
boost::optional<PivotTable> pivotTableOpt = cell.getPivotTable();
if (pivotTableOpt.has_value()) {
    pivotTableOpt.get().remove();
}
```

6.81.2 Метод PivotTable::getSourceRangeAddress

Метод возвращает текстовое представление диапазона исходных данных сводной таблицы.

Пример:

```
Table sheet = document.getBlocks().getTable(0).get();
Cell cell = sheet.getCell("A1");
boost::optional<PivotTable> pivotTableOpt = cell.getPivotTable();
if (pivotTableOpt.has_value()) {
    std::printf("%d", pivotTableOpt.get().getSourceRangeAddress().c_str());
}
```

6.81.3 Метод PivotTable::getSourceRange

Метод возвращает диапазон `CellRange` исходных данных сводной таблицы.

Пример:

```
CellRange sourceRange = pivotTable.getSourceRange();
std::printf("%d %d", sourceRange.getBeginColumn(), sourceRange.getLastColumn());
```

6.81.4 Метод PivotTable::getPivotRange

Метод возвращает диапазон ячеек `CellRange`, в котором размещена сводная таблица.

Пример:

```
CellRange pivotRange = pivotTable.getPivotRange();
std::printf("%d %d", pivotRange.getBeginColumn(), pivotRange.getLastColumn());
```

6.81.5 Метод PivotTable::changeSourceRange

Метод позволяет задать новый диапазон исходных данных сводной таблицы без обновления самой таблицы. Параметр `sourceRange` – строка, представляющая новый диапазон таблицы.

Пример:

```
pivotTable.changeSourceRange("I3:K5");
CellRange sourceRange = pivotTable.getSourceRange();
std::printf("%d %d", sourceRange.getBeginColumn(), sourceRange.getLastColumn());
```

6.81.6 Метод PivotTable::isRowGrandTotalEnabled

Метод возвращает `true`, если разрешено показывать общие итоги для строк.

Пример:

```
std::printf("%d", pivotTable.isRowGrandTotalEnabled());
```

6.81.7 Метод PivotTable::isColumnGrandTotalEnabled

Метод возвращает `true`, если разрешено показывать общие итоги для столбцов.

Пример:

```
std::printf("%d", pivotTable.isColumnGrandTotalEnabled());
```

6.81.8 Метод PivotTable::getPivotTableCaptions

Метод возвращает информацию PivotTableCaptions о всех заголовках сводной таблицы.

Пример:

```
PivotTableCaptions pivotTableCaptions = pivotTable.getPivotTableCaptions();
std::printf("%s", pivotTableCaptions.errorCaption.get().c_str());
std::printf("%s", pivotTableCaptions.emptyCaption.get().c_str());
std::printf("%s", pivotTableCaptions.grandTotalCaption.c_str());
std::printf("%s", pivotTableCaptions.valuesHeaderCaption.c_str());
std::printf("%s", pivotTableCaptions.columnHeaderCaption.c_str());
std::printf("%s", pivotTableCaptions.rowHeaderCaption.c_str());
```

6.81.9 Метод PivotTable::getPivotTableLayoutSettings

Метод возвращает настройки отображения PivotTableLayoutSettings сводной таблицы.

Пример:

```
PivotTableLayoutSettings pivotTableLayoutSettings =
pivotTable.getPivotTableLayoutSettings();
std::printf("%d", pivotTableLayoutSettings.displayFieldCaptions);
std::printf("%d", pivotTableLayoutSettings.indentForCompactLayout);
std::printf("%d", pivotTableLayoutSettings.isMergeAndCenterLabelsEnabled);
std::printf("%d", pivotTableLayoutSettings.pageFieldOrder);
std::printf("%d", pivotTableLayoutSettings.pageFieldWrapCount);
std::printf("%d", pivotTableLayoutSettings.reportLayout);
std::printf("%d", pivotTableLayoutSettings.useGridDropZones);
std::printf("%d", pivotTableLayoutSettings.valueFieldsOrientation);
```

6.81.10 Метод PivotTable::getUnsupportedFeatures

Метод возвращает неподдерживаемые свойства PivotTableUnsupportedFeature сводной таблицы.

Пример:

```
PivotTableUnsupportedFeatures pivotTableUnsupportedFeatures =
pivotTable.getUnsupportedFeatures();
```

```
for (int i = 0; i < pivotTableUnsupportedFeatures.size(); i++) {
    PivotTableUnsupportedFeature feature = pivotTableUnsupportedFeatures[i];
    std::printf("%d", feature);
}
```

6.81.11 Метод PivotTable::getFieldsList

Метод возвращает список PivotTableField всех полей сводной таблицы.

Пример:

```
PivotTableFields pivotTableFields = pivotTable.getFieldsList();
for (int i = 0; i < pivotTableFields.size(); i++) {
    PivotTableField field = pivotTableFields[i];
    std::printf("%s", field.fieldProperties.fieldAlias.get().c_str());
    std::printf("%s", field.fieldProperties.subtotalAlias.get().c_str());
    std::printf("%s", field.fieldProperties.fieldName.c_str());
    std::shared_ptr<Enumerator<PivotTableFieldCategory>> enumerator =
field.fieldCategories.getEnumerator();
    while (enumerator->isValid()) {
        PivotTableFieldCategory category = enumerator->getCurrent();
        std::printf("%d", category);
        enumerator->goToNext();
    }
}
```

6.81.12 Метод PivotTable::getRowFields

Метод возвращает список полей PivotTableCategoryField из области строк.

Пример:

```
PivotTableCategoryFields pivotTableRowFields = pivotTable.getRowFields();
for (int i = 0; i < pivotTableRowFields.size(); i++) {
    PivotTableCategoryField field = pivotTableRowFields[i];
    std::printf("%s", field.fieldProperties.fieldAlias.get().c_str());
    std::printf("%s", field.fieldProperties.subtotalAlias.get().c_str());
    std::printf("%s", field.fieldProperties.fieldName.c_str());
    PivotTableFunctions subtotalFunctions = field.subtotalFunctions;
    for (int j = 0; j < subtotalFunctions.size(); j++) {
        PivotTableFunction subtotalFunction = subtotalFunctions[j];
        std::printf("%d", subtotalFunction);
```

```
    }
}
```

6.81.13 Метод PivotTable::getColumnFields

Метод возвращает список полей PivotTableCategoryField из области колонок.

Пример:

```
boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
PivotTablesManager pivotTablesManager = pivotTablesManagerOpt.get();
PivotTable pivotTable = pivotTablesManager.create(cellRange);
PivotTableCategoryFields columnFields = pivotTable.getColumnFields();
for (int i = 0; i < columnFields.size(); i++) {
    std::printf("%s", columnFields[i].fieldProperties.fieldAlias.get());
    std::printf("%s", columnFields[i].fieldProperties.subtotalAlias.get());
    std::printf("%d", columnFields[i].fieldProperties.fieldName.c_str());
    for (int j = 0; j < columnFields[i].subtotalFunctions.size(); j++) {
        std::printf("%d", columnFields[i].subtotalFunctions[j]);
    }
}
```

6.81.14 Метод PivotTable::getValueFields

Метод возвращает список полей PivotTableValueField из области значений.

Пример:

```
boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
PivotTablesManager pivotTablesManager = pivotTablesManagerOpt.get();
PivotTable pivotTable = pivotTablesManager.create(cellRange);
PivotTableValueFields valueFields = pivotTable.getValueFields();
for (int i = 0; i < valueFields.size(); i++) {
    std::printf("%s", valueFields[i].baseFieldName.c_str());
    std::printf("%d", valueFields[i].cellNumberFormat);
    std::printf("%s", valueFields[i].customFormula.get().c_str());
    std::printf("%d", valueFields[i].totalFunction);
    std::printf("%s", valueFields[i].valueFieldName.c_str());
}
```

6.81.15 Метод PivotTable::getPageFields

Метод возвращает список полей PivotTablePageField из области фильтров.

Пример:

```
boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
PivotTablesManager pivotTablesManager = pivotTablesManagerOpt.get();
PivotTable pivotTable = pivotTablesManager.create(cellRange);
PivotTableCategoryFields pageFields = pivotTable.getPageFields();
for (int i = 0; i < pageFields.size(); i++) {
    std::printf("%s", pageFields[i].fieldProperties.fieldAlias.get());
    std::printf("%s", pageFields[i].fieldProperties.subtotalAlias.get());
    std::printf("%d", pageFields[i].fieldProperties.fieldName.c_str());
}
```

6.81.16 Метод PivotTable::getFieldCategories

Метод возвращает список категорий PivotTableFieldCategories, содержащих заданное поле `fieldName`.

Пример:

```
PivotTableFieldCategories categories = pivotTable.getFieldCategories("Age");
std::shared_ptr<Enumerator<PivotTableFieldCategory>> enumerator =
categories.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFieldCategory category = enumerator->getCurrent();
    std::printf("%d", category);
    enumerator->goToNext();
}
```

6.81.17 Метод PivotTable::getFieldItems

Метод возвращает все элементы PivotTableItems сводной таблицы по заданному имени поля `fieldName`.

Пример:

```
PivotTableItems pivotTableItems = pivotTable.getFieldItems("Age");
std::shared_ptr<Enumerator<PivotTableItem>> enumerator =
pivotTableItems.getEnumerator();
while (enumerator->isValid())
{
```

```
    PivotTableItem pivotTableItem = enumerator->getCurrent();
    std::printf("%s", pivotTableItem.getAlias().get().c_str());
    enumerator->goToNext();
}
```

6.81.18 Метод PivotTable::getFieldItemsByName

Метод возвращает все элементы PivotTableItems из заданного поля fieldName по имени itemName.

Пример:

```
PivotTableItems itemsByName = pivotTable.getFieldItemsByName("Ultimate Question
of Life", "42");
std::shared_ptr<Enumerator<PivotTableItem>> enumerator =
itemsByName.getEnumerator();
while (enumerator->isValid()) {
    PivotTableItem pivotTableItem = enumerator->getCurrent();
    std::printf("%s", pivotTableItem.getAlias().get().c_str());
    enumerator->goToNext();
}
```

6.81.19 Метод PivotTable::getFilter

Метод возвращает фильтр PivotTableFilter по заданному имени поля fieldName.

Пример:

```
boost::optional<PivotTableFilter> filterOpt = pivotTable.getFilter("Age");
if (filterOpt.has_value()) {
    std::printf("%d", filterOpt.get().getCount());
}
```

6.81.20 Метод PivotTable::getFilters

Метод возвращает список фильтров PivotTableFilter сводной таблицы.

Пример:

```
PivotTableFilters filters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
filters.getEnumerator();
while (enumerator->isValid())
{
```

```
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    std::printf("%d", pivotTableFilter.getCount());
    enumerator->goToNext();
}
```

6.81.21 Метод PivotTable::update

Метод обновляет и полностью пересчитывает сводную таблицу, возвращает PivotTableUpdateResult.

Пример:

```
PivotTableUpdateResult updateResult = pivotTable.update();
if (updateResult == PivotTableUpdateResult::FieldAlreadyEnabled) {
    ............
}
```

6.81.22 Метод PivotTable::createPivotTableEditor

Метод возвращает объект PivotTableEditor, который служит для обновления свойств и редактирования сводной таблицы.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor.addField("Age", PivotTableFieldCategory::Rows);
pivotTableEditor.apply();
```

6.82 Класс PivotTableCaptions

Класс PivotTableCaptions хранит все пользовательские заголовки сводной таблицы. Описание полей класса представлено в таблице 43.

Таблица 43 – Описание полей класса PivotTableCaptions

Поле	Описание
PivotTableCaptions.errorCaption	Алиас для значений, которые возвращают ошибку.
PivotTableCaptions.emptyCaption	Алиас для значений, которые возвращают пустое значение.
PivotTableCaptions.grandTotalCaption	Алиас общих итогов.
PivotTableCaptions.valuesHeaderCaption	Алиас поля из области значений; это поле отображается в отчете в случае, если в сводной таблице наличие более двух полей из области

Поле	Описание
	значений, и макет имеют тип ‘outline’ или ‘tabular’.
PivotTableCaptions.rowHeaderCaption	Алиас заголовка строк (виден только при включенном компактном макете, это алиас по умолчанию).
PivotTableCaptions.columnHeaderCaption	Алиас заголовка колонок (виден только при включенном компактном макете, это алиас по умолчанию).

6.83 Класс PivotTableCategoryField

PivotTableCategoryField содержит свойства поля сводной таблицы, использующегося как строка / столбец (см. таблицу 44). Объект может быть получен посредством вызовов PivotTable::getRowFields(), PivotTable::getColumnFields().

Таблица 44 – Описание полей PivotTableCategoryField

Поле	Описание
PivotTableCategoryField .fieldProperties	Свойства поля PivotTableFieldProperties
PivotTableCategoryField .subtotalFunctions	Список функций PivotTableFunction для вычисления подытога

6.84 Класс PivotTableEditor

Предназначен для редактирования сводных таблиц. Возвращается посредством метода PivotTable::createPivotTableEditor().

6.84.1 Метод PivotTableEditor::addField

Метод добавляет новое поле в сводную таблицу, используя параметры: fieldName - имя поля, toCategory - категория поля (тип - PivotTableFieldCategory), index - позиция в категории. Метод возвращает объект PivotTableEditor.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor = pivotTableEditor.addField("CC",
PivotTableFieldCategory::Values);
pivotTableEditor.apply();
```

6.84.2 Метод PivotTableEditor::moveField

Метод перемещает поле между категориями. Параметры: `fieldName` - имя поля, `toCategory` - область, в которую перемещается поле (тип - `PivotTableFieldCategory`), `index` - позиция в новой категориии. Метод возвращает объект `PivotTableEditor`.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor = pivotTableEditor.moveField("CC",
PivotTableFieldCategory::Values, 0);
pivotTableEditor.apply();
```

6.84.3 Метод PivotTableEditor::removeField

Метод удаляет поле из категории. Параметры: `fieldName` - имя поля, `fromCategory` - область, из которой удаляется поле (тип - `PivotTableFieldCategory`). Метод возвращает объект `PivotTableEditor`.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor = pivotTableEditor.removeField("BB",
PivotTableFieldCategory::Values);
pivotTableEditor.apply();
```

6.84.4 Метод PivotTableEditor::reorderField

Метод изменяет позицию поля в пределах категории. Параметры: `fieldName` - имя поля, `category` - область (тип - `PivotTableFieldCategory`), `toIndex` - новая позиция поля. Метод возвращает объект `PivotTableEditor`.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor = pivotTableEditor.reorderField("CC",
PivotTableFieldCategory::Values, 0);
pivotTableEditor.apply();
```

6.84.5 Метод PivotTableEditor::enableField

Метод добавляет поле в область, зависящую от типа поля. Параметр `fieldName` - имя поля. Метод возвращает объект `PivotTableEditor`.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor = pivotTableEditor.enableField("Age");
pivotTableEditor.apply();
```

6.84.6 Метод PivotTableEditor::disableField

Метод удаляет поле из всех областей. Параметр `fieldName` - имя поля (тип - строка). Метод возвращает объект `PivotTableEditor`.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor = pivotTableEditor.disableField("Age");
pivotTableEditor.apply();
```

6.84.7 Метод PivotTableEditor::setSummarizeFunction

Метод задает суммирующую функцию для поля из области значений. Параметр `valueFieldName` - имя поля (тип - строка), `summarizeFunction` – суммирующая функция, тип - `PivotTableFunction`. Метод возвращает объект `PivotTableEditor`.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
PivotTableFunction summarizeFunction = PivotTableFunction::Average;
pivotTableEditor = pivotTableEditor.setSummarizeFunction("CC",
summarizeFunction);
```

6.84.8 Метод PivotTableEditor::setFilter

Метод задает фильтр `PivotTableFilter` сводной таблицы. Если фильтр не может быть применен, вызывается исключение `PivotTableError`. Метод возвращает объект `PivotTableEditor`.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    for (int i = 0; i < pivotTableFilter.getCount(); i++) {
        pivotTableFilter.setHidden(i, false);
```

```
        pivotTableEditor.setFilter(pivotTableFilter);
    }
    enumerator->goToNext();
}
pivotTableEditor.apply();
```

6.84.9 Метод PivotTableEditor::setFilters

Метод задает фильтры PivotTableFilters сводной таблицы. Если какой-то из фильтров не может быть применен, он пропускается. Метод возвращает объект PivotTableEditor.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    for (int i = 0; i < pivotTableFilter.getCount(); i++) {
        pivotTableFilter.setHidden(i, false);
    }
    enumerator->goToNext();
}
pivotTableEditor.setFilters(pivotTableFilters);
pivotTableEditor.apply();
```

6.84.10 Метод PivotTableEditor::setCaptions

Метод задает заголовки сводной таблицы PivotTableCaptions, возвращает объект PivotTableEditor.

Пример:

```
PivotTableCaptions captions = pivotTable.getPivotTableCaptions();
captions.grandTotalCaption = "Общий итог за год";

PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor.setCaptions(captions);
pivotTableEditor.apply();
```

6.84.11 Метод PivotTableEditor::setLayoutSettings

Метод устанавливает настройки отображения PivotTableLayoutSettings сводной таблицы, возвращает объект PivotTableEditor.

Пример:

```
PivotTableLayoutSettings pivotTableLayoutSettings =
pivotTable.getPivotTableLayoutSettings();
pivotTableLayoutSettings.reportLayout = PivotTableReportLayout::Tabular;

PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor = pivotTableEditor.setLayoutSettings(pivotTableLayoutSettings);
pivotTableEditor.apply();
```

6.84.12 Метод PivotTableEditor::setGrandTotalSettings

Метод задает настройки отображения общего итога. Параметры: `isRowGrandTotalEnabled` – показывать общие итоги для строк, `isColGrandTotalEnabled` – показывать общие итоги для столбцов.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor = pivotTableEditor.setGrandTotalSettings(true, true);
pivotTableEditor.apply();
```

6.84.13 Метод PivotTableEditor::apply

Метод обновляет сводную таблицу с заданными свойствами и возвращает результат PivotTableUpdateResult.

Пример:

```
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
if (PivotTableUpdateResult::Success == pivotTableEditor.apply()) {
    std::printf("Successfully applied");
}
```

6.85 Класс PivotTableField

Класс `PivotTableField` содержит свойства полей сводной таблицы (см. таблицу 45). Объект может быть получен посредством вызова PivotTable::getFieldsList().

Таблица 45 – Описание полей класса `PivotTableField`

Поле	Описание
`PivotTableField.fieldProperties`	Свойства полей сводной таблицы `PivotTablFieldProperties`
`PivotTableField.fieldCategories`	Категории полей сводной таблицы `PivotTablFieldCategories`
`PivotTableField.customFormula`	Вычисляемая формула (строка)

6.86 Класс PivotTableFieldCategory

Класс `PivotTableFieldCategory` описывает флаги, которые задают категорию области полей. Описание полей представлено в таблице 46.

Таблица 46 – Описание полей класса `PivotTableFieldCategory`

Поле	Описание
`PivotTableFieldCategory::Pages`	Область фильтров
`PivotTableFieldCategory::Rows`	Область строк
`PivotTableFieldCategory::Columns`	Область колонок
`PivotTableFieldCategory::Values`	Область значений

6.87 Класс PivotTableFieldCategories

Класс обеспечивает доступ к списку категорий поля сводной таблицы. Объект может быть получен посредством использования метода `PivotTable::getFieldCategories()`.

6.87.1 Метод PivotTableFieldCategories::getEnumerator

Метод для перечисления категорий поля `PivotTableFieldCategory`.

Пример:

```
PivotTableFieldCategories categories = pivotTable.getFieldCategories("Age");
std::shared_ptr<Enumerator<PivotTableFieldCategory>> enumerator =
categories.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFieldCategory category = enumerator->getCurrent();
    std::printf("%d", category);
    enumerator->goToNext();
}
```

6.88 Класс PivotTableFieldProperties

`PivotTableFieldProperties` содержит свойства поля `PivotTableField` сводной таблицы (см. таблицу 47).

Таблица 47 – Описание полей класса `PivotTableFieldProperties`

Поле	Описание
`PivotTableFieldProperties.fieldName`	Имя поля
`PivotTableFieldProperties.fieldAlias`	Псевдоним поля (пользовательское имя)
`PivotTableFieldProperties.subtotalAlias`	Псевдоним подытогов конкретного поля

6.89 Класс PivotTableFilter

Позволяет осуществить доступ к списку фильтров таблицы, каждый из которых обладает свойством видимости. При любом изменении фильтров они должны быть применены к сводной таблице посредством использования методов `PivotTableEditor::setFilter()`, `PivotTableEditor::setFilters()`.

Пример:

```
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    for (int i = 0; i < pivotTableFilter.getCount(); i++) {
        pivotTableFilter.setHidden(i, false);
    }
    enumerator->goToNext();
}
PivotTableEditor pivotTableEditor = pivotTable.createPivotTableEditor();
pivotTableEditor.setFilters(pivotTableFilters);
pivotTableEditor.apply();
```

6.89.1 Метод PivotTableFilter::getFieldName

Возвращает имя поля, с которым ассоциирован фильтр.

Пример:

```
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
```

```
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    std::printf("%s", pivotTableFilter.getFieldName().c_str());
    enumerator->goToNext();
}
```

6.89.2 Метод PivotTableFilter::getCount

Возвращает количество фильтруемых полей.

Пример:

```
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    std::printf("%d", pivotTableFilter.getCount());
    enumerator->goToNext();
}
```

6.89.3 Метод PivotTableFilter::getName

Возвращает имя поля для заданного индекса.

Пример:

```
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    for (int i = 0; i < pivotTableFilter.getCount(); i++) {
        std::printf("%s", pivotTableFilter.getName(i).c_str());
    }
    enumerator->goToNext();
}
```

6.89.4 Метод PivotTableFilter::isHidden

Возвращает видимость поля для заданного индекса `itemIndex`. Если `true`, то поле скрыто.

Пример:

```
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    for (int i = 0; i < pivotTableFilter.getCount(); i++) {
        std::printf("%d", pivotTableFilter.isHidden(i));
    }
    enumerator->goToNext();
}
```

6.89.5 Метод PivotTableFilter::setHidden

Устанавливает видимость поля для заданного индекса. Параметры: `itemName` – индекс поля, `hidden` – видимость (`true` – поле скрыто).

Пример:

```
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    for (int i = 0; i < pivotTableFilter.getCount(); i++) {
        pivotTableFilter.setHidden(i, false);
    }
    enumerator->goToNext();
}
```

6.90 Класс PivotTableFilters

Класс обеспечивает доступ к списку фильтров. Для получения объекта PivotTableFilters используется метод `PivotTable::getFilters()`.

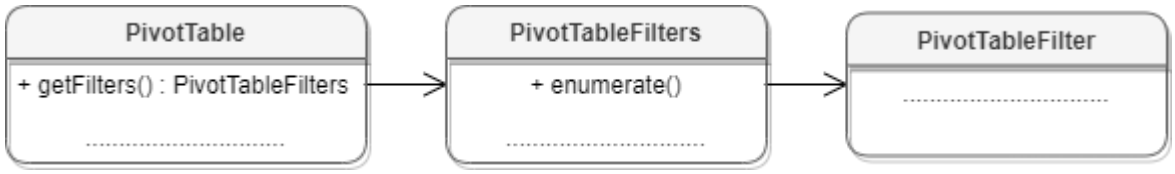


Рисунок 33 – Объектная модель классов для работы с фильтрами

6.90.1 Метод PivotTableFilters::getEnumerator

Метод используется для доступа к коллекции фильтров (см. PivotTableFilter).

Пример:

```
PivotTableFilters pivotTableFilters = pivotTable.getFilters();
std::shared_ptr<Enumerator<PivotTableFilter>> enumerator =
pivotTableFilters.getEnumerator();
while (enumerator->isValid()) {
    PivotTableFilter pivotTableFilter = enumerator->getCurrent();
    std::printf("%s", pivotTableFilter.getFieldName().c_str());
    enumerator->goToNext();
}
```

6.91 Класс PivotTableFunction

Класс PivotTableFunction описывает функции, которые могут быть использованы в сводных таблицах. Описание полей представлено в таблице 48. Объект используется в качестве поля subtotalFunctions класса PivotTableCategoryField.

Таблица 48 – Описание полей класса PivotTableFunction

Поле	Описание
PivotTableFunction::Auto	Автозаполнение
PivotTableFunction::Sum	Суммирует все числовые данные
PivotTableFunction::Count	Количество всех ячеек
PivotTableFunction::CountNums	Количество числовых ячеек
PivotTableFunction::Average	Среднее значение
PivotTableFunction::Max	Наибольшее значение
PivotTableFunction::Min	Наименьшее значение
PivotTableFunction::Product	Произведение всех ячеек
PivotTableFunction::StdDeviation	Стандартное смещенное отклонение
PivotTableFunction::StdDeviationPopulation	Стандартное несмещенное отклонение
PivotTableFunction::Variance	Смещенная дисперсия
PivotTableFunction::VariancePopulation	Несмещенная дисперсия

6.92 Класс PivotTableItem

`PivotTableItem` описывает элемент сводной таблицы (см. Рисунок 34). См. пример в главе `PivotTableItems::enumerate`.

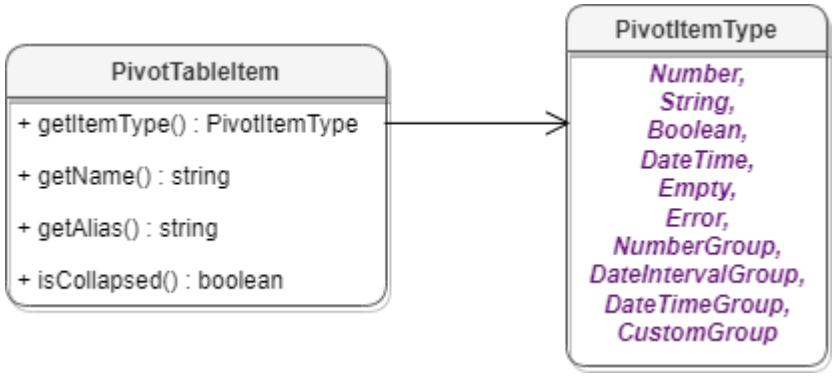


Рисунок 34 – Класс `PivotTableItem`

6.92.1 Метод PivotTableItem::getName

Метод возвращает имя элемента сводной таблицы, тип - строка. См. пример в главе `PivotTableItems::getEnumerator`.

6.92.2 Метод PivotTableItem::getAlias

Метод возвращает псевдоним элемента (идентификатор, созданный пользователем), тип - строка. См. пример в главе `PivotTableItems::getEnumerator`.

6.92.3 Метод PivotTableItem::getItemType

Метод возвращает тип `PivotTableItemType` элемента сводной таблицы. См. пример в главе `PivotTableItems::getEnumerator`.

6.92.4 Метод PivotTableItem::isCollapsed

Метод возвращает `true`, если элемент сводной таблицы свернут. См. пример в главе `PivotTableItems::getEnumerator`.

6.93 Класс PivotTableItems

Класс обеспечивает доступ к списку элементов сводной таблицы (см. Рисунок 35).

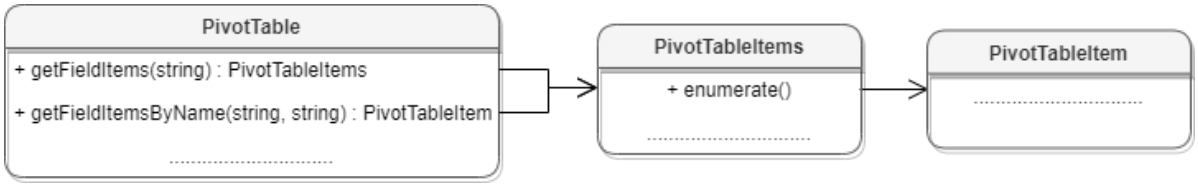


Рисунок 35 – Объектная модель классов для работы с элементами сводных таблиц

6.93.1 Метод PivotTableItems::getEnumerator

Используется для перечисления элементов сводной таблицы.

Пример:

```
PivotTableItems pivotTableItems = pivotTable.getFieldItems("Age");
std::shared_ptr<Enumerator<PivotTableItem>> enumerator =
pivotTableItems.getEnumerator();
while (enumerator->isValid()) {
    PivotTableItem pivotTableItem = enumerator->getCurrent();
    boost::optional<std::string> alias = pivotTableItem.getAlias();
    if (alias.has_value()) {
        std::printf("%s", alias.get().c_str());
    }
    std::printf("%s", pivotTableItem.getName().c_str());
    std::printf("%d", pivotTableItem.getItemType());
    std::printf("%d", pivotTableItem.isCollapsed());
    enumerator->goToNext();
}
```

6.94 Класс PivotTableItemType

Класс `PivotTableItemType` содержит возможные типы элементов сводной таблицы. Описание полей представлено в таблице 49.

Таблица 49 – Описание полей класса `PivotTableItemType`

Поле	Описание
`PivotTableItemType::Number`	Числовой
`PivotTableItemType::String`	Строковый
`PivotTableItemType::Boolean`	Логический
`PivotTableItemType::DateTime`	Дата / время
`PivotTableItemType::Empty`	Пустой тип
`PivotTableItemType::Error`	Ошибка
`PivotTableItemType::NumberGroup`	Интервальная группировка
`PivotTableItemType::DateIntervalGroup`	Интервальная группировка по датам
`PivotTableItemType::DateTimeGroup`	Группировка по дате / времени
`PivotTableItemType::CustomGroup`	Пользовательская (произвольная) группировка

Пример:

```
PivotTableItems pivotTableItems = pivotTable.getFieldItems("Age");
std::shared_ptr<Enumerator<PivotTableItem>> enumerator =
pivotTableItems.getEnumerator();
while (enumerator->isValid()) {
    PivotTableItem pivotTableItem = enumerator->getCurrent();
    PivotTableItemType pivotTableItemType = pivotTableItem.getItemType();
    std::printf("%d", pivotTableItemType);
    enumerator->goToNext();
}
```

6.95 Класс PivotTableLayoutSettings

Класс `PivotTableLayoutSettings` содержит настройки отображения сводной таблицы. Данный объект может быть получен в результате вызова `PivotTable::getPivotTableLayoutSettings()` и установлен методом `PivotTableEditor::setLayoutSettings()`. Описание полей класса представлено в таблице 50.

Таблица 50 – Описание полей класса PivotTableLayoutSettings

Поле	Описание
PivotTableLayoutSettings.reportLayout	Настройка вида макета сводной таблицы (PivotTableReportLayout: компактный, табличный, структурный).
PivotTableLayoutSettings.valueFieldsOrientation	Настраивает положение значений в случае, если в сводной таблице более двух полей значений. Тип - ValueFieldsOrientation.
PivotTableLayoutSettings.pageFieldOrder	Настройка порядка полей фильтров (PageFieldOrder: вниз, затем поперек или сначала поперек, потом вниз).
PivotTableLayoutSettings.indentForCompactLayout	Размер отступа для полей в области строк в компактном макете (режим иерархии в случае наличия более двух полей).
PivotTableLayoutSettings.pageFieldWrapCount	Настройка связана с pageFieldOrder, она показывает через сколько полей будет совершено указанное действие (перенос на следующую строку и т.д).
PivotTableLayoutSettings.isMergeAndCenterLabelsEnabled	Настройка позволяет объединить ячейки заголовков.
PivotTableLayoutSettings.useGridDropZones	Флаг, отвечающий за отображение классического вида (как в Excel 2003). Влияет только на расположение полей в отчете.
PivotTableLayoutSettings.displayFieldCaptions	Флаг, отвечающий за отображение заголовков полей.

6.96 Класс PivotTablePageField

Содержит свойства поля из области фильтров (см. таблицу 51). Объект может быть получен посредством вызова PivotTable::getPageFields().

Таблица 51 – Описание полей класса PivotTablePageField

Поле	Описание
PivotTablePageField.fieldProperties	Свойства поля PivotTableFieldProperties

6.97 Класс PivotTableReportLayout

Класс PivotTableReportLayout описывает внешний вид отчетов сводной таблицы. Используется в качестве поля reportLayout класса PivotTableLayoutSettings. Описание полей PivotTableReportLayout представлено в таблице 52.

Таблица 52 – Описание полей класса `PivotTableReportLayout`

Поле	Описание
`PivotTableReportLayout::Compact`	Компактный вид
`PivotTableReportLayout::Tabular`	Табличный вид
`PivotTableReportLayout::Outline`	Структурный вид

6.98 Класс PivotTablesManager

Класс `PivotTablesManager` используется для создания сводных таблиц, содержит метод `create()`. Может быть получена вызовом `Document::getPivotTablesManager()`.

Пример:

```
boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
```

6.98.1 Метод PivotTablesManager:create

Метод создает сводную таблицу `PivotTable` на основе диапазона исходных данных `CellRange`.

Если местоположение не задано, создается новый лист (таблица), и сводная таблица будет расположена по умолчанию.

Пример:

```
Table sheet = document.getBlocks().getTable("Лист1").get();
CellRange cellRange = sheet.getCellRange("B3:C4");

boost::optional<PivotTablesManager> pivotTablesManagerOpt =
document.getPivotTablesManager();
PivotTablesManager pivotTablesManager = pivotTablesManagerOpt.get();
PivotTable pivotTable = pivotTablesManager.create(cellRange);
```

6.99 Класс PivotTableUnsupportedFeature

Класс `PivotTableUnsupportedFeature` описывает неподдерживаемую функциональность сводных таблиц. Получение неподдерживаемой функциональности сводных таблиц описано в `PivotTable::getUnsupportedFeatures()`. Описание полей класса представлено в таблице 53.

Таблица 53 – Описание полей класса `PivotTableUnsupportedFeature`

Поле	Описание
`PivotTableUnsupportedFeature::CalculatedField`	Вычисляемые поля
`PivotTableUnsupportedFeature::CalculatedItem`	Вычисляемые элементы
`PivotTableUnsupportedFeature::CollapsedValues`	Свернутые поля
`PivotTableUnsupportedFeature::ShowDataAs`	Вычисления (Show data как в MS Excel)

6.100 Класс PivotTableUpdateResult

В таблице 54 приведены константы, которые соответствуют возможным результатам обновления сводной таблицы (см. методы `PivotTable::update()`, `PivotTableEditor:apply()`).

Таблица 54 – Результаты обновления сводной таблицы

Наименование константы	Описание
`PivotTableUpdateResult::Success`	Успешное обновление таблицы.
`PivotTableUpdateResult::NoPivotTable`	Сводная таблица не найдена.
`PivotTableUpdateResult::NoSuchFieldInCategory`	Не найдено поле в категории.
`PivotTableUpdateResult::NoSuchFieldInPivotTable`	Не найдено поле в сводной таблице.
`PivotTableUpdateResult::InvalidIndex`	Ошибка в индексе.
`PivotTableUpdateResult::FieldAlreadyEnabled`	Поле уже существует.
`PivotTableUpdateResult::MovingFieldToTheSameCategoryForbidden`	Попытка перемещения поля в рамках текущей категории .
`PivotTableUpdateResult::InvalidFunction`	Неправильная функция.
`PivotTableUpdateResult::InvalidCategory`	Неправильная область.
`PivotTableUpdateResult::InvalidDataSourceRange`	Ошибка диапазона исходных данных.
`PivotTableUpdateResult::NoDataRowsInDataSource`	В исходных данных нет строк с данными.

PivotTableUpdateResult::EmptyDataSourceHeaders	Пустые заголовки исходных данных.
PivotTableUpdateResult::NoReferenceUnderDefine	Попытка обновить или создать сводную таблицу на именованном диапазоне который не содержит ссылку, а содержит константу.
PivotTableUpdateResult::NoSuchItem	Элемент не найден.
PivotTableUpdateResult::CannotExpandCollapseLeafItem	Не удается раскрыть свернутый элемент.
PivotTableUpdateResult::AnotherPivotInsideDataSource	Найдена другая сводная таблица в этом же диапазоне.
PivotTableUpdateResult::Canceled	Обновление сводной таблицы отменено.

6.101 Класс PivotTableValueField

PivotTableValueField содержит свойства поля сводной таблицы, использующегося как значение столбец (см. таблицу 55). Таблица может быть получена посредством вызова PivotTable::getValueFields().

Таблица 55 – Описание полей класса PivotTableValueField

Поле	Описание
PivotTableValueField.baseFieldName	Оригинальное поле на основе которого было создано данное поле, тип - строка.
PivotTableValueField.valueFieldName	Автоматический уникальный псевдоним такой как “Sum of %имя поля%“, тип – строка.
PivotTableValueField.cellNumberFormat	Числовой формат типа CellFormat для конкретного поля значений.
PivotTableValueField.totalFunction	Агрегирующая функция PivotTableFunction поля значений (SUM, COUNT, MAX и т.д).
PivotTableValueField.customFormula	Вычисляемая формула для поля значений, тип - строка.

6.102 Класс Position

Класс Position представляет местоположение в текстовом документе. Используется для обозначения начала и конца диапазона Range.

6.102.1 Метод Position:insertText

Метод предназначен для вставки текстовой строки в заданное местоположение в документе.

Пример:

```
Range range = document.getRange();
Position startPosition = range.getBegin();
startPosition.insertText("Текст в начале строки");
```

6.102.2 Метод Position:insertTable

Метод предназначен для вставки таблицы с заданным числом строк и столбцов в заданное местоположение в документе. Возвращает объект таблицы.

Следует учитывать, что при вставке таблицы к ее имени автоматически добавляется порядковый номер, начинающийся с единицы. Таким образом, вызов

```
Table t = position.insertTable(3, 3, "Table")
```

приведет к созданию в текстовом документе таблицы с именем «Table1».

Пример вставки таблицы в текстовый документ:

```
Position beginPosition = document.getRange().getBegin();
Table table = beginPosition.insertTable(3, 3, "Table");
```

В табличном документе данный метод используется для вставки нового рабочего листа.

Пример вставки нового листа в табличный документ:

```
Position endPosition = document.getRange().getEnd();
Table table = endPosition.insertTable(3, 3, "Table");
```

6.102.3 Метод Position:insertPageBreak

Метод предназначен для вставки разрыва страницы в указанную позицию в документе.

Пример:

```
Range range = document.getRange();
Position endPos = range.getEnd();
endPos.insertPageBreak();
```

6.102.4 Метод Position:insertLineBreak

Метод предназначен для вставки перевода строки.

Пример:

```
Range range = document.getRange();
Position endPos = range.getEnd();
endPos.insertLineBreak();
```

6.102.5 Метод Position:insertBookmark

Вставляет закладку с наименованием в текущую позицию.

Пример:

```
Range range = document.getRange();
Position endPos = range.getEnd();
endPos.insertBookmark("Bookmark");
```

6.102.6 Метод Position:insertSectionBreak

Вставляет разрыв раздела в текущую позицию.

Примеры:

```
Range range = document.getRange();
Position endPos = range.getEnd();
endPos.insertSectionBreak();
```

```
uint8_t NextPage = 0;
uint8_t Continious = 1;
uint8_t EvenPage = 2;
uint8_t OddPage = 3;

Range range = document.getRange();
Position endPos = range.getEnd();
endPos.insertSectionBreak((SectionBreakType)NextPage);
```

6.102.7 Метод Position:insertImage

Вставляет изображение из файла в текущую позицию.

Вызов:

```
insertImage( url, size )
```

Параметры:

– `url` – полный путь к файлу;

– `size` – геометрические размеры изображения для вставки.

Пример:

```
document.getRange().getBegin().insertImage("C://Tmp//123.jpg",
Size<float>(100.0, 100.0));
```

6.102.8 Метод Position:removeBackward

Метод удаляет `count` объектов (символов, картинок и т.д.) до текущей позиции.

Пример:

```
Position endPosition = document.getRange().getEnd();
endPosition.removeBackward(3);
```

6.102.9 Метод Position:removeForward

Метод удаляет `count` объектов (символов, картинок и т.д.) после текущей позиции.

Пример:

```
Position beginPosition = document.getRange().getBegin();
beginPosition.removeForward(3);
```

6.103 Класс PrintingScope

Класс `PrintingScope` содержит настройки для экспорта табличных документов. Используется в поле `printingScope` класса `WorkbookExportSettings`.

Позволяет создать области печати следующих типов:

- выбранная область печати либо весь документ (см. `PrintingScope::Type`);
- указанный диапазон ячеек (см. `CellRangePosition`).

Примеры:

```
// по умолчанию - PrintingScope.Type::PrintArea
printingScope = PrintingScope();
```

```
// область печати
printingScope = PrintingScope(PrintingScope.Type::PrintArea);
```

```
// диапазон ячеек
cellRangePosition = CellRangePosition(0, 0, 5, 5);
printingScope = PrintingScope(cellRangePosition);
```

6.103.1 Метод PrintingScope::getCellRange

Метод возвращает диапазон ячеек таблицы (см. `CellRangePosition`).

6.103.2 Метод PrintingScope::usePrintArea

Метод возвращает `true`, если область печати должна использоваться во время печати, экспорта и т. д.

6.103.3 Тип PrintingScope.Type

Варианты выбора диапазона страниц для экспорта и печати представлены в таблице 56. Используется в `PrintingScope`

Таблица 56 – Диапазон страниц для экспорта и печати

Константа	Описание
`PrintingScope.Type::PrintArea`	Выбранная область печати (по умолчанию)
`PrintingScope.Type::WholeSheet`	Печать всего документа

6.104 Класс Range

Класс `Range` предоставляет доступ к диапазону документа. На рисунке 36 изображена объектная модель классов, относящихся к работе с диапазонами.

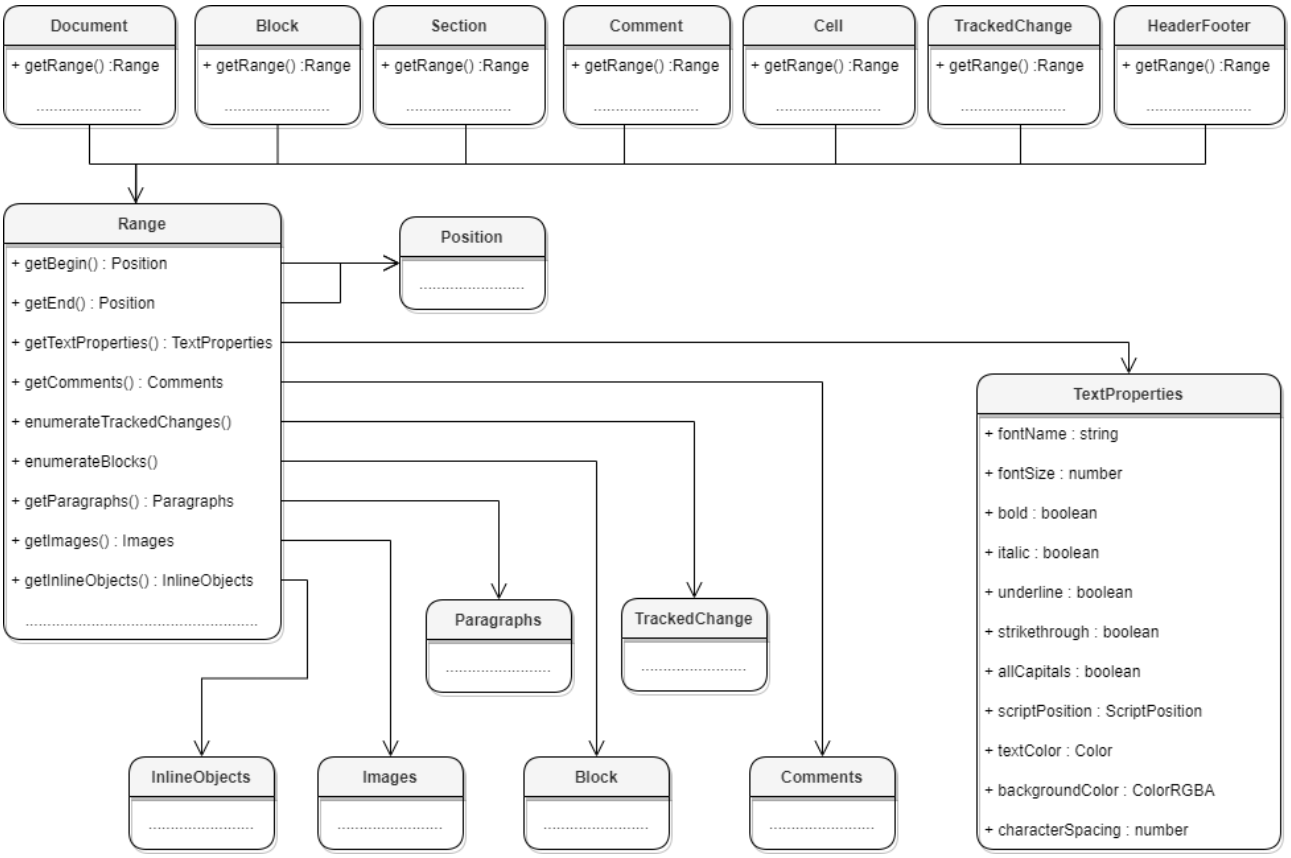


Рисунок 36 – Объектная модель для работы с классом `Range`

Варианты получения диапазона для текстового документа:

```
// диапазон всего документа
Range range = document.getRange();

// диапазон блока
boost::optional<Block> blockOpt = document.getBlocks().getBlock(0);
if (blockOpt.has_value()) {
    Range firstBlockRange = blockOpt.get().getRange();
}

// диапазон секций
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    Section section = enumerator->getCurrent();
    Range sectionRange = section.getRange();
    enumerator->goToNext();
}
```

```
// диапазон комментариев
Comments comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
while (enumerator->isValid()) {
    Comment comment = enumerator->getCurrent();
    Range commentRange = comment.getRange();
    enumerator->goToNext();
}

// диапазон ячейки
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Cell cell = tableOpt.get().getCell("B2");
    Range cellRange = cell.getRange();
}

// диапазон верхних колонтитулов
boost::optional<Block> blockOpt = document.getBlocks().getBlock(0);
if (blockOpt.has_value()) {
    Section section = blockOpt.get().getSection();
    HeadersFooters headers = section.getHeaders();
    std::shared_ptr<Enumerator<HeaderFooter>> enumerator =
headers.getEnumerator();
    while (enumerator->isValid()) {
        HeaderFooter header = enumerator->getCurrent();
        Range headerRange = header.getRange();
        enumerator->goToNext();
    }
}

// диапазон отслеживаемых изменений
std::shared_ptr<Enumerator<TrackedChange>> enumerator =
document.getRange().getTrackedChangesEnumerator();
while (enumerator->isValid()) {
    TrackedChange trackedChange = enumerator->getCurrent();
    Range commentRange = trackedChange.getRange();
```

```
    enumerator->goToNext();
}
```

6.104.1 Метод Range::getBegin

Метод возвращает позицию в начале диапазона.

Пример для текстового документа:

```
Position beginDocPos = document.getRange().getBegin();
beginDocPos.insertText("API");
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    Position beginDocPos = cellRange.getBegin();
    beginDocPos.insertText("API");
}
```

6.104.2 Метод Range::getEnd

Метод возвращает позицию в конце диапазона, не включая последний символ paragraph mark.

Пример для текстового документа:

```
Position endDocPos = document.getRange().getEnd();
endDocPos.insertText("API");
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    Position endDocPos = cellRange.getEnd();
    endDocPos.insertText("API");
}
```

6.104.3 Метод Range::extractText

Метод возвращает содержимое фрагмента в виде строки текста. Находящиеся внутри области изображения, таблицы и другие объекты игнорируются.

Пример для текстового документа:

```
Range range = document.getRange();
std::printf("%s", range.extractText().c_str());
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    std::printf("%s", cellRange.extractText().c_str());
}
```

6.104.4 Метод Range::removeContent

Метод полностью удаляет содержимое диапазона.

Пример для текстового документа:

```
Range range = document.getRange();
range.removeContent();
std::printf("%s", range.extractText().c_str());
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    cellRange.removeContent();
    std::printf("%s", cellRange.extractText().c_str());
}
```

6.104.5 Метод Range::lockContent

Метод запрещает изменения содержимого диапазона.

Внимание ! Метод может быть использован только в текстовых документах.

Пример для текстового документа:

```
Range range = document.getRange();
range.lockContent();
```

Пример для таблицы внутри текстового документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    cellRange.lockContent();
}
```

6.104.6 Метод Range::unlockContent

Метод разрешает изменения содержимого диапазона.

Внимание ! Метод может быть использован только в текстовых документах.

Пример для текстового документа:

```
Range range = document.getRange();
range.unlockContent();
```

Пример для таблицы внутри текстового документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    cellRange.unlockContent();
}
```

6.104.7 Метод Range::isContentLocked

Метод возвращает значение `true`, если изменения содержимого диапазона запрещены.

Пример для текстового документа:

```
Range range = document.getRange();
if (range.isContentLocked()) {
```

```
    std::printf("Документ содержит заблокированное содержимое");
}
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    if (cellRange.isContentLocked()) {
        std::printf("Ячейка содержит заблокированное содержимое");
    }
}
```

6.104.8 Метод Range::replaceText

Метод заменяет содержимое фрагмента на указанный текст.

Пример для текстового документа:

```
Range range = document.getRange();
range.replaceText("New text");
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    cellRange.replaceText("New text");
}
```

6.104.9 Метод Range::getTextProperties

Метод возвращает объект с текущими настройками форматирования для фрагмента текстового документа. Описание настроек форматирования осуществляется с помощью объекта TextProperties.

Пример для текстового документа:

```
Range range = document.getRange();
TextProperties textProperties = range.getTextProperties();
boost::optional<std::string> fontNameOpt = textProperties.fontName;
```

```
if (fontNameOpt.has_value()) {
    std::printf("%s", fontNameOpt.get().c_str());
}
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    TextProperties textProperties = cellRange.getTextProperties();
    boost::optional<std::string> fontNameOpt = textProperties.fontName;
    if (fontNameOpt.has_value()) {
        std::printf("%s", fontNameOpt.get().c_str());
    }
}
```

6.104.10 Метод Range::setTextProperties

Метод применяет настройки форматирования `TextProperties` для диапазона.

Пример для текстового документа:

```
Range range = document.getRange();
TextProperties textProperties = range.getTextProperties();
textProperties.fontName = "Arial";
range.setTextProperties(textProperties);
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    TextProperties textProperties = cellRange.getTextProperties();
    textProperties.fontName = "Arial";
    cellRange.setTextProperties(textProperties);
}
```

6.104.11 Метод Range::getBlocksEnumerator

Предоставляет возможность итерации по блокам.

Пример для текстового документа:

```
Range range = document.getRange();
std::shared_ptr<Enumerator<Block>> enumerator = range.getBlocksEnumerator();
while (enumerator->isValid()) {
    Block block = enumerator->getCurrent();
    std::printf("%s", block.getRange().extractText().c_str());
    enumerator->goToNext();
}
```

Пример для табличного документа:

```
boost::optional<Table> tableOpt = document.getBlocks().getTable(0);
if (tableOpt.has_value()) {
    Table table = tableOpt.get();
    Cell cell = table.getCell("B2");
    Range cellRange = cell.getRange();
    std::shared_ptr<Enumerator<Block>> enumerator =
cellRange.getBlocksEnumerator();
    while (enumerator->isValid()) {
        Block block = enumerator->getCurrent();
        std::printf("%s", block.getRange().extractText().c_str());
        enumerator->goToNext();
    }
}
```

6.104.12 Метод Range::getTrackedChangesEnumerator

Предоставляет возможность итерации по отслеживаемым изменениям TrackedChange. Метод может быть использован только в текстовых документах.

Пример:

```
Range range = document.getRange();
std::shared_ptr<Enumerator<TrackedChange>> enumerator =
range.getTrackedChangesEnumerator();
while (enumerator->isValid()) {
    TrackedChange trackedChange = enumerator->getCurrent();
    std::printf("%s", trackedChange.getRange().extractText().c_str());
```

```
    enumerator->goToNext();
}
```

6.104.13 Метод Range::getComments

Обеспечивает доступ к комментариям в диапазоне.

Комментарии, примененные к одному и тому же диапазону, упорядочиваются по датам. Если дат нет, то порядок комментариев не определен.

Пример:

```
auto comments = document.getRange().getComments();
std::shared_ptr<Enumerator<Comment>> enumerator = comments.getEnumerator();
while (enumerator->isValid()) {
   Comment comment = enumerator->getCurrent();
   std::printf("%s", comment.getText().get().c_str());
   enumerator->goToNext();
}
```

6.104.14 Метод Range::getParagraphs

Обеспечивает доступ к абзацам Paragraphs в диапазоне.

Пример для текстового документа:

```
Range range = document.getRange();
Paragraphs paragraphs = range.getParagraphs();
std::shared_ptr<Enumerator<Paragraph>> enumerator = paragraphs.getEnumerator();
while (enumerator->isValid()) {
    Paragraph paragraph = enumerator->getCurrent();
    ......................
    enumerator->goToNext();
}
```

Пример для табличного документа:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("B2");
Range range = cell.getRange();
Paragraphs paragraphs = range.getParagraphs();
while (enumerator->isValid()) {
    Paragraph paragraph = enumerator->getCurrent();
    ......................
```

```
    enumerator->goToNext();
}
```

6.104.15 Метод Range::getImages

Обеспечивает доступ к изображениям (Image) в диапазоне.

Примеры:

```
CO::API::Document::Images images = document.getRange().getImages();
std::shared_ptr<Enumerator<Image>> enumerator = images.getEnumerator();
while (enumerator->isValid()) {
    Image image = enumerator->getCurrent();
    Frame frame = image.getFrame();
    boost::optional < CO::API::Size<float>> dimensionsOpt =
frame.getDimensions();
    if (dimensionsOpt.has_value()) {
        Size<float> size = dimensionsOpt.get();
        std::printf("%f", size.width);
    }
    enumerator->goToNext();
}
```

6.104.16 Метод Range::getInlineObjects

Обеспечивает доступ к перечислению InlineObjects графических объектов диапазона.

Пример:

```
Range range = document.getRange();
InlineObjects inlineObjects = range.getInlineObjects();
std::shared_ptr<Enumerator<InlineObject>> enumerator =
inlineObjects.getEnumerator();
while (enumerator->isValid()) {
    InlineObject inlineObject = enumerator->getCurrent();
    boost::optional<Size<float>> dimensions =
inlineObject.getFrame().getDimensions();
    if (dimensions.has_value()) {
        std::printf("%d", dimensions.get().height);
    }
}
```

6.105 Класс RangeBorders

Класс `RangeBorders` оставлен для совместимости. Вместо него необходимо использовать класс `Borders`.

6.106 Класс SaveDocumentSettings

Класс `SaveDocumentSettings` предоставляет настройки, используемые для сохранения документа в файл (см. `document::saveAs()`). Описание полей класса `SaveDocumentSettings` представлено в таблице 57.

Таблица 57 – Описание полей класса `SaveDocumentSettings`

Поле	Описание
`SaveDocumentSettings.documentFormat`	Формат документа `DocumentFormat`.
`SaveDocumentSettings.documentType`	Тип документа `DocumentType`.
`SaveDocumentSettings.documentPassword`	Пароль для защиты электронного документа от несанкционированного доступа.
`SaveDocumentSettings.isTemplate`	Флаг, обозначающий, что документ должен быть сохранен как шаблон.
`SaveDocumentSettings.dsvSettings`	Структура `DSVSettings`, необходимая для сохранения в формате DSV.

6.107 Класс ScientificCellFormatting

Класс содержит параметры для экспоненциального формата ячеек таблицы. Используется в качестве аргумента метода `Cell::setFormat()`. Описание полей класса `ScientificCellFormatting` представлено в таблице 58.

Таблица 58 – Описание полей класса `ScientificCellFormatting`

Поле	Описание
`ScientificCellFormatting.decimalPlaces`	Количество десятичных позиций
`ScientificCellFormatting.minExponentDigits`	Минимальное количество позиций экспоненты

Пример:

```
Table firstSheet = document.getBlocks().getTable("Лист1").get();
Cell cell = firstSheet.getCell("A2");

ScientificCellFormatting cellFormat = ScientificCellFormatting();
cellFormat.decimalPlaces = 2;
```

```
cellFormat.minExponentDigits = 3;


cell.setFormat(cellFormat);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.108 Класс Scripts

Класс `Scripts` предоставляет доступ к списку макрокоманд документа. Коллекцию макрокоманд `Scripts` можно получить из документа посредством вызова метода `Document::getScripts()`.

Пример:

```
std::shared_ptr<Scripts> scripts = document.getScripts();
if (scripts) {
    std::shared_ptr<Enumerator<std::shared_ptr<Script>>> enumerator = scripts-
>getEnumerator();
    while (enumerator->isValid()) {
        enumerator->goToNext();
    }
}
```

6.108.1 Метод Scripts::getScript

Метод возвращает объект класса `Script`, описывающий макрокоманду. В качестве аргумента используется имя макрокоманды.

Пример:

```
std::shared_ptr<Scripts> scripts = document.getScripts();
if (scripts) {
    std::shared_ptr<Script> script = scripts->getScript("ScriptName");
    if (script) {
    }
}
```

6.108.2 Метод Scripts::setScript

Метод добавляет макрокоманду в текущий документ. Если макрокоманда с таким именем уже существует, будет обновлено ее содержимое.

Пример:

```
std::shared_ptr<Scripts> scripts = document.getScripts();
if (scripts) {
```

```
    std::string scriptName = "Enumerate scripts for document";
    std::string scriptCode = "local scripts = document:getScripts()\nfor script
in scripts:enumerate() do\nprint(script:getName())\nend";
    scripts->setScript(scriptName, scriptCode);
    std::shared_ptr<Script> script = scripts->getScript(scriptName);
    if (script) {
    }
}
```

6.108.3 Метод Scripts::removeScript

Метод удаляет макрокоманду из текущего документа. В качестве аргумента используется имя макрокоманды.

Пример:

```
std::string scriptName = "Enumerate scripts for document";
scripts->removeScript(scriptName);
std::shared_ptr<Script> script = scripts->getScript(scriptName);
if (!script) {
    std::printf("Script was removed");
}
```

6.108.4 Метод Scripts::getEnumerator

Метод возвращает коллекцию макрокоманд для их дальнейшего перечисления.

Пример:

```
std::shared_ptr<Scripts> scripts = document.getScripts();
if (scripts) {
    std::shared_ptr<Enumerator<std::shared_ptr<Script>>> enumerator = scripts-
>getEnumerator();
    while (enumerator->isValid()) {
        std::shared_ptr<Script> script = enumerator->getCurrent();
        if (script) {
            std::printf(script->getName().c_str());
        }
        enumerator->goToNext();
    }
}
```

6.109 Класс Script

Класс `Script` предназначен для управления отдельной макрокомандой. Содержит поля `Name` и `Body`.

6.109.1 Метод Script::getName

Метод возвращает имя макрокоманды.

Пример:

```
std::shared_ptr<Script> script = enumerator->getCurrent();
if (script) {
    std::printf(script->getName().c_str());
}
```

6.109.2 Метод Script::setName

Метод устанавливает имя для макрокоманды.

Пример:

```
std::shared_ptr<Script> script = enumerator->getCurrent();
if (script) {
    script->setName("New Script Name");
    std::printf(script->getName().c_str());
}
```

6.109.3 Метод Script::getBody

Метод возвращает текст макрокоманды в виде строки.

Пример:

```
std::shared_ptr<Script> script = enumerator->getCurrent();
if (script) {
    std::printf(script->getBody().c_str());
}
```

6.109.4 Метод Script::setBody

Метод устанавливает текст макрокоманды, полностью заменяя уже имеющийся текст.

Пример:

```
std::shared_ptr<Script> script = enumerator->getCurrent();
if (script) {
    script->setBody("local scripts = document:getScripts()\nfor script in
scripts : enumerate() do\nprint(script:getName())\nend");
```

```
    std::printf(script->getName().c_str());
}
```

6.110 Клссс Scripting

Объект класса `Scripting::Scripting` может быть получен путем вызова `Scripting::createScripting()` и содержит метод `runScript`, который используется для запуска макрокоманды.

6.110.1 Метод Scripting::createScripting

Функция `Scripting::createScripting` возвращает объект класса `Scripting`. В качестве параметра используется текущий документ.

Пример:

```
std::shared_ptr<Scripting::Scripting> scripting =
Scripting::createScripting(document);
if (scripting) {
}
```

6.110.2 Метод Scripting::runScript

Запускает макрокоманду с указанным именем. В случае невозможности запуска макрокоманды вызывает исключение `ScriptExecutionError`.

Пример:

```
std::shared_ptr<Scripting::Scripting> scripting =
Scripting::createScripting(document);
if (scripting) {
    try {
        scripting->runScript("ScriptName");
    }
    catch (const ScriptExecutionError& error) {
        std::printf("%s", error.what());
    }
}
```

6.111 Класс ScriptPosition

Варианты представления текста в виде надстрочных или подстрочных знаков при работе в текстовом редакторе представлены в таблице 59. Используется в качестве поля `scriptPosition` класса `TextProperties`.

Таблица 59 – Типы надстрочного и подстрочного форматирования

Наименование константы	Описание
`ScriptPosition::SuperScript`	Надстрочный знак (верхний индекс)
`ScriptPosition::SubScript`	Подстрочный знак (нижний индекс)
`ScriptPosition::NoramlScript`	Без указания индекса

Пример:

```
TextProperties textProperties = TextProperties();
textProperties.scriptPosition = ScriptPosition::NormalScript;
document.getRange().setTextProperties(textProperties);
```

6.112 Класс Search

Класс `Search` предоставляет доступ к механизму поиска и замены фрагментов документа, открытого в редакторе текста или таблиц.

6.112.1 Метод Search::findText

Метод выполняет поиск строки без учета регистра во всем документе или выбранном диапазоне документа. Результат возвращается в виде диапазона `Range`, содержащего искомый фрагмент.

Если строка не обнаружена, возвращается пустой диапазон.

Примеры:

```
// Поиск по всему документу
std::shared_ptr<Search> search = createSearch(document);
std::shared_ptr<Enumerator<Range>> searchResult = search->findText("API");
while (searchResult->isValid()) {
    Range searchRange = searchResult->getCurrent();
    std::printf("%s", searchRange.extractText().c_str());
    searchResult->goToNext();
}
```

```
// Поиск только в диапазоне блока
Range firstBlockRange = document.getBlocks().getBlock(0).get().getRange();
std::shared_ptr<Search> search = createSearch(document);
std::shared_ptr<Enumerator<Range>> searchResult = search->findText("API",
firstBlockRange);
.....................................................................
```

6.113 Класс Section

Класс `Section` представляет собой раздел в документе.

6.113.1 Метод Section::setPageProperties

Метод устанавливает параметры `PageProperties` страниц, находящихся в разделе.

Пример:

```
boost::optional<Block> blockOpt = document.getBlocks().getBlock(0);
if (blockOpt.has_value()) {
    Section section = blockOpt.get().getSection();
    PageProperties pageProperties = section.getPageProperties();
    pageProperties.height = 100;
    pageProperties.width = 200;
    section.setPageProperties(pageProperties);
}
```

6.113.2 Метод Section::getPageProperties

Метод возвращает параметры страниц раздела `PageProperties`.

Пример:

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    Section section = enumerator->getCurrent();
    PageProperties pageProperties = section.getPageProperties();
    std::printf("%d, %d", pageProperties.height, pageProperties.width);
    enumerator->goToNext();
}
```

6.113.3 Метод Section::setPageOrientation

Метод задает ориентацию страниц раздела.

Пример:

```
Section section = enumerator->getCurrent();
section.setPageOrientation(PageOrientation::Landscape);
boost::optional<PageOrientation> pageOrientationOpt =
section.getPageOrientation();
if (pageOrientationOpt.has_value()) {
    PageOrientation pageOrientation = pageOrientationOpt.get();
```

```
    std::printf("%s", pageOrientation == PageOrientation::Landscape ?
"Landscape" : "Portrait");
}
```

6.113.4 Метод Section::getPageOrientation

Метод возвращает ориентацию страниц раздела.

Пример:

```
boost::optional<PageOrientation> pageOrientationOpt =
section.getPageOrientation();
if (pageOrientationOpt.has_value()) {
    PageOrientation pageOrientation = pageOrientationOpt.get();
    std::printf("%s", pageOrientation == PageOrientation::Landscape ?
"Landscape" : "Portrait");
}
```

6.113.5 Метод Section::getRange

Метод возвращает диапазон Range в документе, соответствующий данному разделу.

Пример:

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    Section section = enumerator->getCurrent();
    std::printf("%s", section.getRange().extractText().c_str());
    enumerator->goToNext();
}
```

6.113.6 Метод Section::getHeaders

Метод возвращает коллекцию HeadersFooters верхних колонтитулов данного раздела.

Пример:

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    Section section = enumerator->getCurrent();
    HeadersFooters headers = section.getHeaders();
    ..................................................
```

```
    enumerator->goToNext();
}
```

6.113.7 Метод Section::getFooters

Метод возвращает коллекцию HeadersFooters нижних колонтитулов данного раздела.

Пример:

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    Section section = enumerator->getCurrent();
    HeadersFooters footers = section.getFooters();
    ..................................................
    enumerator->goToNext();
}
```

6.114 Класс Sections

Класс Sections используется для доступа к коллекции секций документа. Описание секции см. в разделе Section.

6.114.1 Метод Sections::getEnumerator

Метод позволяет перечислить коллекцию секций документа.

Пример:

```
Sections sections = document.getSections();
std::shared_ptr<Enumerator<Section>> enumerator = sections.getEnumerator();
while (enumerator->isValid()) {
    enumerator->goToNext();
}
```

6.115 Класс Shape

Класс Shape представляет собой фигуру, содержит методы для установки и получения свойств ShapeProperties.

6.115.1 Метод Shape::getShapeProperties

Метод возвращает свойства фигуры `ShapeProperties`.

Пример:

```
boost::optional<Shape> shapeOpt = document.getBlocks().getShape(0);
if (shapeOpt.has_value()) {
    ShapeProperties shapeProperties = shapeOpt.get().getShapeProperties();
    std::printf("%d", shapeProperties.verticalAlignment);
}
```

6.115.2 Метод Shape::setShapeProperties

Метод устанавливает свойства фигуры `ShapeProperties`.

Пример:

```
boost::optional<Shape> shapeOpt = document.getBlocks().getShape(0);
if (shapeOpt.has_value()) {
    ShapeProperties shapeProperties = shapeOpt.get().getShapeProperties();
    shapeProperties.verticalAlignment = VerticalAlignment::Center;
    shapeOpt.get().setShapeProperties(shapeProperties);
}
```

6.116 Класс ShapeProperties

Класс описывает свойства фигуры и содержит следующие поля:

– `verticalAlignment` - вертикальное выравнивание, тип `VerticalAlignment`;

– `borderProperties` - свойства границ фигуры, тип `LineProperties`;

– `fill` - свойства заполнения фигуры, тип `Fill`;

– `shapeTextLayout` - свойства текста внутри фигуры, тип `ShapeTextLayout`.

6.117 Класс ShapeTextLayout

Класс `ShapeTextLayout` описывает свойства текста, находящегося внутри фигуры. Описание полей представлено в таблице 60. Используется в качестве поля класса `ShapeProperties`.

Таблица 60 – Описание полей класса `ShapeTextLayout`

Поле	Описание
`ShapeTextLayout::DoNotAutoFit`	Размещение текста в фигуре по умолчанию

Поле	Описание
`ShapeTextLayout::FitShapeExtentToText`	Расширение фигуры под текст
`ShapeTextLayout::FitTextToShape`	Заполнение фигуры текстом

6.118 Класс Table

Класс `Table` предоставляет доступ к листу в табличном документе или таблице в составе текстового документа (см. Рисунок 37).

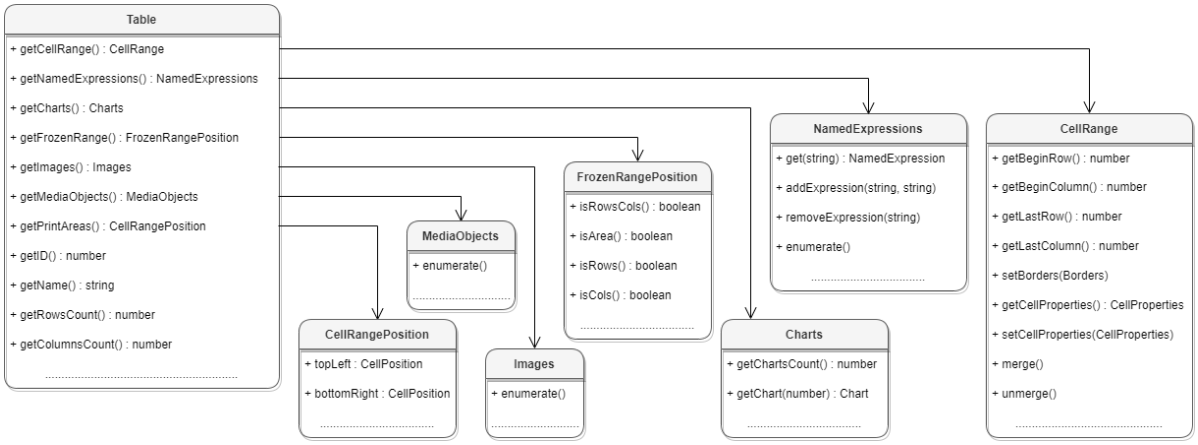


Рисунок 37 – Структура полей класса `Table`

6.118.1 Метод Table::setName

Метод задает имя таблицы. В случае с табличным документом это имя будет являться заголовком листа документа. Данное значение должно быть уникальным, т.к. может использоваться для ссылки на таблицу, например, из формул.

Пример:

```
Table table = document.getBlocks().getTable("Лист1").get();
table.setName("Table1");
```

Для текстовых документов использование данного метода также допустимо, наименование таблицы нигде не отображается, но в дальнейшем его можно использовать для доступа к таблице по имени.

Пример:

```
std::string tableName = "Table1";
Table table = document.getBlocks().getTable(0).get();
table.setName(tableName);
table = document.getBlocks().getTable(tableName).get();
```

6.118.2 Метод Table::getName

Метод позволяет получить наименование листа табличного документа.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
std::printf("%s", table.getName().c_str());
```

6.118.3 Метод Table::getRowsCount

Метод позволяет получить количество строк таблицы.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
std::printf("%d", table.getRowsCount());
```

6.118.4 Метод Table::getColumnsCount

Метод позволяет получить количество столбцов таблицы.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
std::printf("%d", table.getColumnsCount());
```

6.118.5 Метод Table::getCell

Метод позволяет получить доступ к отдельной ячейке таблицы. В качестве аргумента может выступать текстовое представление адреса ячейки, либо экземпляр класса `CellPosition`.

Примеры:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("B2");
std::printf("%s", cell.getFormattedValue().c_str());
```

```
CellPosition cellPosition = CellPosition(2, 1);
Cell cell = table.getCell(cellPosition);
std::printf("%s", cell.getFormattedValue().c_str());
```

6.118.6 Метод Table::getCellRange

Метод позволяет получить доступ к диапазону ячеек класса `CellRange`. В качестве аргумента может использоваться строка, описывающая диапазон (`"A1:C4"`), либо объект типа `CellRangePosition`.

Примеры:

```
Table table = document.getBlocks().getTable(0).get();
CellRange cellRange = table.getCellRange("A1:C4");
std::shared_ptr<Enumerator<Cell>> enumerator = cellRange.getEnumerator();
while (enumerator->isValid()) {
    Cell cell = enumerator->getCurrent();
    std::printf("%s", cell.getFormattedValue().c_str());
    enumerator->goToNext();
}
```

```
Table table = document.getBlocks().getTable(0).get();
CellRangePosition cellRangePosition = CellRangePosition(0, 0, 5, 5);
cellRange = table.getCellRange(cellRangePosition);
```

6.118.7 Метод Table::insertColumnAfter

Метод предназначен для вставки нового столбца после указанной позиции в таблице.

Вызов:

```
insertColumnAfter( columnIndex, copyColumnStyle, columnsCount )
```

Параметры:

- `columnIndex` – индекс столбца в таблице, после которого производится вставка. Индексация столбцов начинается с нуля.
- `copyColumnStyle` – флаг наследования стиля. Если этот параметр установлен в значение `true`, то новый столбец наследует настройки форматирования столбца с индексом `columnIndex`. Если параметр `copyColumnStyle` установлен в значение `false`, то настройки форматирования не копируются. Значение по умолчанию `true`.
- `columnsCount` – количество вставляемых столбцов. Значение по умолчанию 1.

Пример:

```
// Создать в документе новую таблицу 2x2
Table table = document.getRange().getBegin().insertTable(2, 2, "NewTable");
// Добавление двух столбцов в середину таблицы, без наследования настроек
форматирования
table.insertColumnAfter(0, false, 2);
```

6.118.8 Метод Table::insertColumnBefore

Метод предназначен для вставки нового столбца до указанной позиции в таблице.

Вызов:

```
insertColumnBefore( columnIndex, copyColumnStyle, columnsCount )
```

Параметры:

- `columnIndex` – индекс столбца в таблице, перед которым производится вставка. Индексация столбцов начинается с нуля.
- `copyColumnStyle` – флаг наследования стиля. Если этот параметр установлен в значение `true`, то новый столбец наследует настройки форматирования столбца с индексом `columnIndex`. Если параметр `copyColumnStyle` установлен в значение `false`, то настройки форматирования не копируются. Значение по умолчанию `true`.
- `columnsCount` – количество вставляемых столбцов. Значение по умолчанию 1.

Пример:

```
// Создать в документе новую таблицу 2x2
Table table = document.getRange().getBegin().insertTable(2, 2, "NewTable");
// Добавление двух столбцов в середину таблицы, без наследования настроек
форматирования
table.insertColumnBefore(1, false, 2);
```

6.118.9 Метод Table::insertRowAfter

Метод предназначен для вставки новой строки после указанной позиции в таблице.

Вызов:

```
insertRowAfter( rowIndex, copyRowStyle, rowsCount )
```

Параметры:

- `rowIndex` – индекс строки в таблице, после которой производится вставка. Индексация строк начинается с нуля.
- `copyRowStyle` – флаг наследования стиля. Если этот параметр установлен в значение `true`, то новая строка наследует настройки форматирования строки с индексом `rowIndex`. Если параметр `copyRowStyle` установлен в значение `false`, то настройки форматирования не копируются. Значение по умолчанию `true`.
- `rowsCount` – количество вставляемых строк. Значение по умолчанию 1.

Пример:

```
// Создать в документе новую таблицу 2x2
Table table = document.getRange().getBegin().insertTable(2, 2, "NewTable");

// Добавление двух строк в середину таблицы, без наследования настроек форматирования
table.insertRowAfter(0, false, 2);
```

6.118.10 Метод Table::insertRowBefore

Метод предназначен для вставки новой строки до указанной позиции в таблице.

Вызов:

```
insertRowBefore( rowIndex, copyRowStyle, rowsCount )
```

Параметры:

- `rowIndex` – индекс строки в таблице, перед которой производится вставка. Индексация строк начинается с нуля.
- `copyRowStyle` – флаг наследования стиля. Если этот параметр установлен в значение `true`, то новая строка наследует настройки форматирования строки с индексом `rowIndex`. Если параметр `copyRowStyle` установлен в значение `false`, то настройки форматирования не копируются. Значение по умолчанию `true`.
- `rowsCount` – количество вставляемых строк. Значение по умолчанию 1.

Пример:

```
// Создать в документе новую таблицу 2x2
Table table = document.getRange().getBegin().insertTable(2, 2, "NewTable");

// Добавление двух строк в середину таблицы, без наследования настроек форматирования
table.insertRowBefore(1, false, 2);
```

6.118.11 Метод Table::removeColumn

Метод предназначен для удаления столбца таблицы, начиная с заданного индекса.

Вызов:

```
removeColumn(columnIndex, columnsCount)
```

Параметры:

- `columnIndex` – индекс столбца, начиная с которого будет удалено заданное количество столбцов. Индексация столбцов начинается с нуля.
- `columnsCount` – количество столбцов для удаления. Значение по умолчанию 1.

6.118.12 Метод Table::removeRow

Метод предназначен для удаления строки таблицы, начиная с заданного индекса.

Вызов:

```
removeRow(rowIndex, rowsCount)
```

Параметры:

- `rowIndex` – индекс строки, начиная с которого будет удалено `rowsCount` строк. Индексация строк начинается с нуля.
- `rowsCount` – количество строк для удаления. Значение по умолчанию 1.

6.118.13 Метод Table:groupRows

Метод предназначен для группировки строк таблицы, начиная с заданного индекса. Индексация строк начинается с нуля.

Вызов:

```
groupRows(rowIndex, rowsCount)
```

Параметры:

- `rowIndex` – индекс строки, начиная с которого будет начата группировка строк;
- `rowsCount` – количество строк для группировки.

6.118.14 Метод Table:groupColumns

Метод предназначен для группировки столбцов таблицы, начиная с заданного индекса. Индексация столбцов начинается с нуля.

Вызов:

```
groupColumns(columnIndex, columnsCount)
```

Параметры:

- `columnIndex` – индекс столбца, начиная с которого будет начата группировка столбцов;
- `columnsCount` – количество столбцов для группировки.

6.118.15 Метод Table:ungroupRows

Метод предназначен для разгруппировки строк таблицы, начиная с заданного индекса. Индексация строк начинается с нуля.

Вызов:

```
ungroupRows(rowIndex, rowsCount)
```


Параметры:

- `rowIndex` – индекс строки, начиная с которого будет начата разгруппировка строк;
- `rowsCount` – количество строк для разгруппировки.

6.118.16 Метод Table:clearRowGroups

Метод предназначен для очистки группированных строк таблицы, начиная с заданного индекса. Индексация строк начинается с нуля.

Вызов:

```
clearRowGroups(rowIndex, rowCount)
```

Параметры:

- `rowIndex` – индекс строки, начиная с которой будет начата очистка групп;
- `rowCount` – количество строк для очистки групп.

6.118.17 Метод Table:ungroupColumns

Метод предназначен для разгруппировки столбцов таблицы, начиная с заданного индекса. Индексация столбцов начинается с нуля.

Вызов:

```
ungroupColumns(columnIndex, columnsCount)
```

Параметры:

- `columnIndex` – индекс столбца, начиная с которого будет начата разгруппировка столбцов;
- `columnsCount` – количество столбцов для разгруппировки.

6.118.18 Метод Table:clearColumnGroups

Метод предназначен для очистки группированных столбцов таблицы, начиная с заданного индекса. Индексация столбцов начинается с нуля.

Вызов:

```
clearColumnGroups(columnIndex, columnsCount)
```

Параметры:

- `columnIndex` – индекс столбца, начиная с которого будет начата очистка групп;
- `columnsCount` – количество столбцов для очистки групп.

6.118.19 Метод Table:setColumnsVisible

Метод `Table::setColumnsVisible` позволяет задавать видимость столбцов, начиная с заданного индекса. Индексация столбцов начинается с нуля.

Вызов:

```
setColumnsVisible(first, columnsCount, visible)
```

Параметрs:

`first` – начальный индекс;

`columnsCount` – количество столбцов;

`visible` – видимость.

6.118.20 Метод Table:setRowsVisible

Метод `Table::setRowsVisible` позволяет задавать видимость строк, начиная с заданного индекса. Индексация строк начинается с нуля.

Вызов:

```
setRowsVisible(first, rowsCount, visible)
```

Параметрs:

`first` – начальный индекс;

`columnsCount` – количество строк;

`visible` – видимость.

6.118.21 Метод Table::setColumnWidth

Метод устанавливает ширину столбца таблицы в пунктах (1/72 дюйма).

Вызов:

```
setColumnWidth( columnIndex, width )
```

Параметры:

- `columnIndex` – индекс столбца в таблице, для которого устанавливается значение ширины. Индексация столбцов начинается с нуля.
- `width` – ширина столбца в пунктах (1/72 дюйма).

Пример:

```
Table table = document.getRange().getBegin().insertTable(2, 2, "NewTable");

// Установить ширину столбца в 400 pt
table.setColumnWidth(1, 400);
```

6.118.22 Метод Table::setRowHeight

Метод устанавливает высоту строки таблицы в пунктах (1/72 дюйма).

Вызов:

```
setRowHeight(rowIndex, height)
```

Параметры:

- `rowIndex` – индекс строки в таблице, для которой устанавливается значение высоты. Индексация строк начинается с нуля.
- `height` – высота строки в пунктах (1/72 дюйма).
- `rowHeightRule` – точность значения (`RowHeightRule::Exact` – точно, `RowHeightRule::AtLeast` – не меньше).

Пример:

```
Table table = document.getRange().getBegin().insertTable(2, 2, "NewTable");

// Установить высоту строки в 100 pt
table.setRowHeight(1, 100, RowHeightRule::Exact);
```

6.118.23 Метод Table::duplicate

Для создания копии листа в табличном документе используется метод `duplicate`. Созданная копия листа размещается после копируемого листа. Метод может быть использован только в табличном документе.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
table.duplicate();
```

6.118.24 Метод Table::remove

Для удаления таблицы в текстовом документе или листа в табличном документе используется метод `remove()`.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
table.remove();
```

6.118.25 Метод Table::moveTo

Для перемещения листа таблицы по указанному индексу в табличном документе используется метод `moveTo`. Указанный индекс должен быть меньше или равен количеству

листов в документе. Индексация листов начинается с нуля. Метод может быть использован только в табличном документе.

Пример:

```
// В табличном документе два листа с индексами 0 и 1.
// Поменяем их местами.
Table table = document.getBlocks().getTable(0).get();
table.moveTo(1);
```

6.118.26 Метод Table::setShowZeroValue

Для упрощения чтения таблицы нулевые значения ячеек могут быть скрыты. Для управления скрытием/показом ячеек используется метод `setShowZeroValue`.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
table.setShowZeroValue(true);
```

6.118.27 Метод Table::getShowZeroValue

Для проверки режима отображения нулевых значений ячеек используется метод `getShowZeroValue`.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
table:setShowZeroValue(false)
std::printf("%d", table:getShowZeroValue());
```

6.118.28 Метод Table::setVisible

Метод управляет видимостью листа таблицы. Используется только в табличном документе.

Вызов:

```
setVisible( visible )
```

Параметр:

`visible` – параметр, задающий видимость листа. Если значение параметра `visible` равно `true`, то лист таблицы отображается в редакторе таблиц.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
table:setVisible(false)
```

6.118.29 Метод Table::isVisible

Метод возвращает значение true, если лист таблицы в табличном документе отображается в редакторе таблиц.

Пример:

```
Table table = document.getBlocks().getTable(0).get();
if (!table.isVisible()) {
    table.setVisible(true);
}
```

6.118.30 Операция ==

Метод используется для определения эквивалентности экземпляров класса.

Пример:

```
boost::optional<Table> tableOpt1 = document.getBlocks().getTable(0);
boost::optional<Table> tableOpt2 = document.getBlocks().getTable(1);
if (tableOpt1.has_value() && tableOpt2.has_value() && (tableOpt1.get() ==
tableOpt2.get())) {
    std::printf("Tables are equal");
}
```

6.118.31 Операция !=

Метод используется для определения неэквивалентности экземпляров класса.

Пример:

```
boost::optional<Table> tableOpt1 = document.getBlocks().getTable(0);
boost::optional<Table> tableOpt2 = document.getBlocks().getTable(1);
if (tableOpt1.has_value() && tableOpt2.has_value() && (tableOpt1.get() !=
tableOpt2.get())) {
    std::printf("Tables are not equal");
}
```

6.118.32 Метод Table::setPrintArea

Метод служит для установки и сброса области печати CellRangePosition.

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
firstSheet.setPrintArea(CellRangePosition(0, 0, 5, 5));
```

6.118.33 Метод Table::getCharts

Для получения списка диаграмм (Charts) таблицы используется метод

`Table:getCharts`.

Пример:

```
std::shared_ptr<Enumerator<Table>> tablesEnumerator =
document.getBlocks().getTablesEnumerator();
while (tablesEnumerator->isValid()) {
    Table table = tablesEnumerator->getCurrent();
    std::printf("%d", table.getCharts().getChartsCount());
    tablesEnumerator->goToNext();
}
```

6.118.34 Метод Table::getNamedExpressions

Для получения списка именованных диапазонов `NamedExpressions` используется метод `Table:getNamedExpressions()`.

Пример:

```
std::shared_ptr<Enumerator<Table>> blocksEnumerator =
document.getBlocks().getTablesEnumerator();
while (blocksEnumerator->isValid()) {
    Table table = blocksEnumerator->getCurrent();
    NamedExpressions namedExpressions = table.getNamedExpressions();
    std::shared_ptr<Enumerator<NamedExpression>> enumerator =
namedExpressions.getEnumerator();
    ..................................
    blocksEnumerator->goToNext();
}
```

6.119 Класс TableRangeInfo

Класс `TableRangeInfo` описывает диапазон ячеек таблицы.

Описание полей класса `TableRangeInfo` представлено в таблице 61.

Таблица 61 – Поля класса `TableRangeInfo`

Поле	Тип	Описание
`tableRange`	`CellRangePosition`	Диапазон ячеек

Пример:

```
Table table = document.getBlocks().getTable(0).get();
Charts charts = table.getCharts();
ChartRangeInfo rangeInfo = charts.getChart(0).getRange(0);
TableRangeInfo cellRangePosition = rangeInfo.tableRangeInfo;
```

```
CellRangePosition tableRange = cellRangePosition.tableRange;
std::printf("topLeft=%d, %d", tableRange.topLeft.row,
tableRange.topLeft.column);
std::printf("bottomRight=%d, %d", tableRange.bottomRight.row,
tableRange.bottomRight.column);
```

6.120 Класс TextAnchoredPosition

Класс `TextAnchoredPosition` (см. Рисунок 38) представляет позицию объекта на странице текстового документа. Используется в качестве позиции в таблице `DocumentAPI.Frame`. Примеры использования см. в разделах `Frame:setPosition()`, `Frame:getPosition()`.

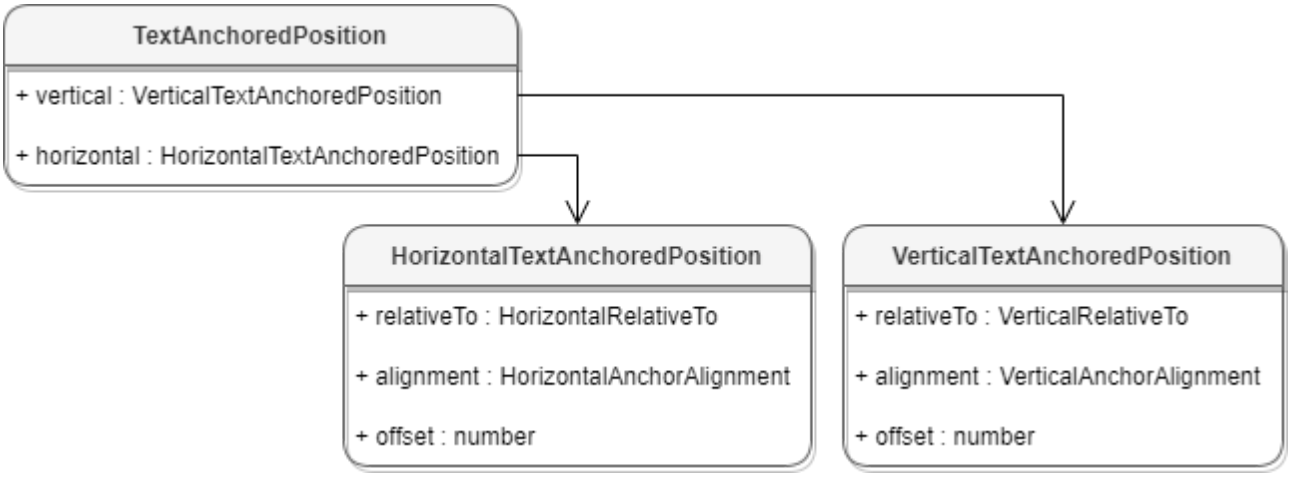


Рисунок 38 – Поля таблицы `DocumentAPI.TextAnchoredPosition`

Описание полей представлено в таблице 62.

Таблица 62 – Описание полей класса `TextAnchoredPosition`

Поле	Описание
`TextAnchoredPosition::horizontal`	Позиция по горизонтали `HorizontalTexAnchoredPosition`
`TextAnchoredPosition::vertical`	Позиция по вертикали `VerticalTextAnchoredPosition`

Пример:

```
TextAnchoredPosition textAnchoredPosition = TextAnchoredPosition();

textAnchoredPosition.horizontal =
boost::optional
```

```
<HorizontalTextAnchoredPosition>(HorizontalRelativeTo::Character);
HorizontalTextAnchoredPosition horzPosition =
textAnchoredPosition.horizontal.get();
horzPosition.offset = 10.0;
horzPosition.relativeTo = HorizontalRelativeTo::ColumnInsideMargin;
horzPosition.alignment = HorizontalAnchorAlignment::Center;

textAnchoredPosition.vertical =
boost::optional<VerticalTextAnchoredPosition>(VerticalRelativeTo::Character);
VerticalTextAnchoredPosition vertTextAnchoredPos =
textAnchoredPosition.vertical.get();
vertTextAnchoredPos.offset = 15.0;
vertTextAnchoredPos.relativeTo = VerticalRelativeTo::PageBottomMargin;
vertTextAnchoredPos.alignment = VerticalAnchorAlignment::Inside;

frame.setPosition(textAnchoredPosition);
```

6.120.1 Оператор ==

Метод используется для определения эквивалентности двух объектов `TextAnchoredPosition`.

6.120.2 Оператор !=

Метод используется для определения неэквивалентности двух объектов `TextAnchoredPosition`.

6.121 Класс TextExportSettings

Класс `TextExportSettings` предоставляет настройки, необходимые для экспорта текстовых документов (см. `document::exportAs()`). Поле объекта `TextExportSettings.pageNumbers` является экземпляром класса `PageNumbers`, в котором содержатся настройки страниц для экспорта текстовых документов.

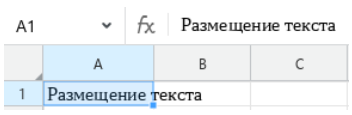
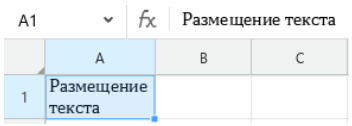
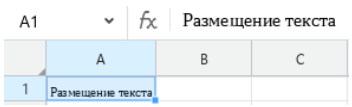
Пример:

```
TextExportSettings textExportSettings = TextExportSettings();
textExportSettings.pageNumbers = PageNumbers(PageParity::Even);
document.exportAs(filePath, ExportFormat::PDFA1, textExportSettings);
```

6.122 Класс TextLayout

В таблице 63 приведены варианты размещения текста в ячейках таблицы. Данное значение используется в поле `textLayout` таблицы `CellProperties`.

Таблица 63 – Варианты размещения текста в ячейках таблицы

Наименование константы	Описание	Отображение
`TextLayout::SingleLine`	Текст располагается в одну строку с наложением на соседние ячейки.	
`TextLayout::WrapByWords`	Текст внутри ячейки переносится по словам. Высота ряда увеличивается чтобы разместить текст полностью.	
`TextLayout::ShrinkSizeToFitWidth`	Текст располагается в одну линию, отображение масштабируется таким образом, чтобы полностью разместиться в ячейке без изменения ее размера. Размер шрифта не изменяется, данная настройка влияет только на отображение содержимого ячейки таблицы.	

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A3");
CellProperties cellProps = cell.getCellProperties();
cellProps.textLayout = TextLayout::ShrinkSizeToFitWidth;
cell.setCellProperties(cellProps);
```

6.123 Класс TextOrientation

Класс `TextOrientation` предоставляет доступ к свойствам ориентации текста в ячейке, фигуре и т. д (см. `CellProperties`).

Пример:

```
Table firstSheet = tableDoc.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A3");

CellProperties cellProps = cell.getCellProperties();
cellProps.textOrientation = TextOrientation(45.5);
cell.setCellProperties(cellProps);
```

6.123.1 Метод TextOrientation:getAngle

Возвращает угол направления текста в ячейке. Значение угла указывается в градусах.

Пример:

```
CellProperties cellProps = cell.getCellProperties();
CellProps.textOrientation = TextOrientation(45);
if (cellProps.textOrientation.has_value()) {
    std::printf("%f", cellProps.textOrientation.get().getAngle());
}
```

6.124 Класс TextProperties

Класс `TextProperties` содержит поля, задающие параметры текста. На рисунке 39 изображена объектная модель класса `TextProperties`.

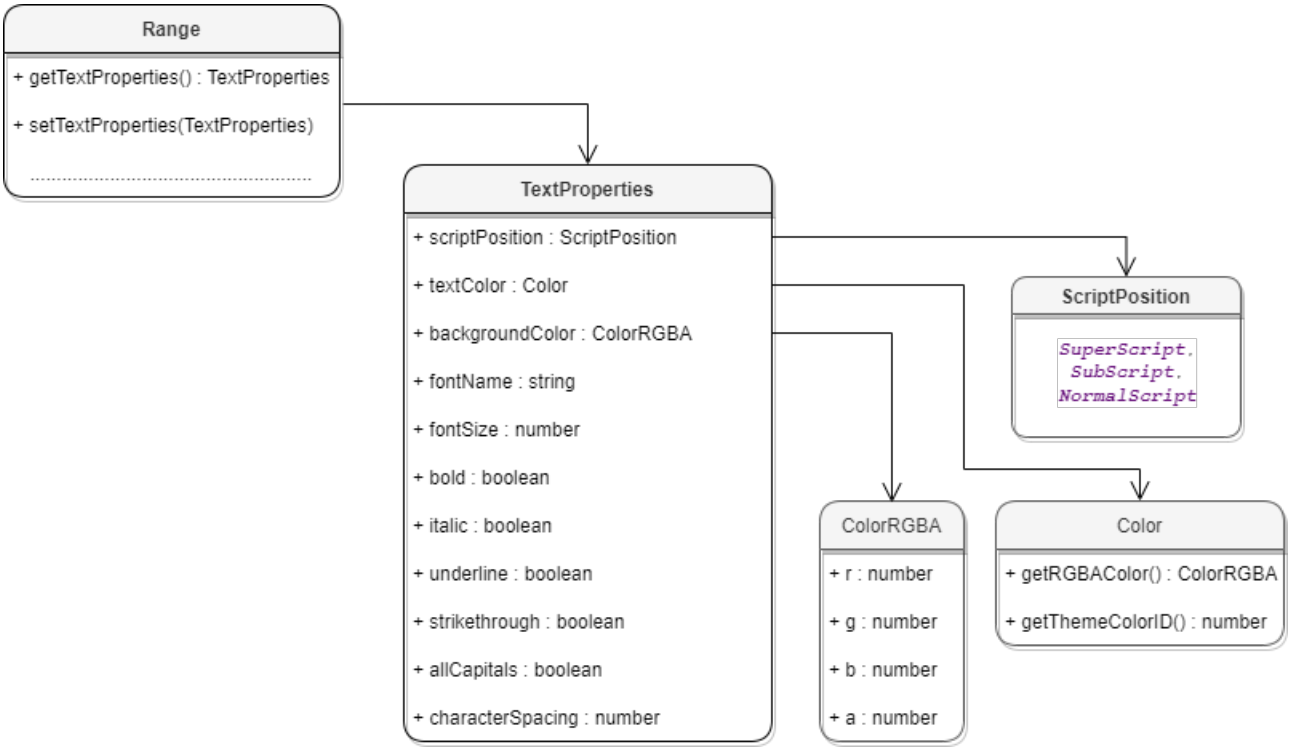


Рисунок 39 – Объектная модель для работы с классом `TextProperties`

Класс `TextProperties` предназначена для форматирования текста. Описание полей класса `TextProperties` представлено в таблице 64. Свойства `TextProperties` применяются к диапазону текста `Range` (методы `Range::getTextProperties()`, `Range::setTextProperties()`).

Таблица 64 – Описание полей класса `TextProperties`

Поле	Тип	Описание
`TextProperties.fontName`	Строковое	Наименование шрифта, использованного для оформления фрагмента документа.
`TextProperties.fontSize`	Числовое	Размер шрифта, использованного для оформления фрагмента документа.
`TextProperties.bold`	Логическое	Значение `true` устанавливает жирное начертание для указанного фрагмента текста.
`TextProperties.italic`	Логическое	Значение `true` устанавливает начертание курсивом для указанного фрагмента текста.
`TextProperties.underline`	Логическое	Значение `true` устанавливает подчеркивание для указанного фрагмента текста.
`TextProperties.strikethrough`	Логическое	Значение `true` устанавливает начертание «зачеркнутый» для указанного фрагмента текста.
`TextProperties.allCapitals`	Логическое	Значение `true` устанавливает все буквы указанного фрагмента текста как прописные. Значение `false` устанавливает все буквы указанного фрагмента текста как строчные.
`TextProperties.scriptPosition`	`ScriptPosition`	Устанавливает отображение символа в виде надстрочного, подстрочного знака или в нормальном режиме.
`TextProperties.textColor`	`Color`	Цвет указанного фрагмента документа.
`TextProperties.backgroundColor`	`ColorRGBA`	Цвет фона указанного фрагмента документа.
`TextProperties.characterSpacing`	Числовое	Размер межсимвольного интервала.

Пример:

```
TextProperties textProperties = TextProperties();
textProperties.fontName = "XO Oriel";
textProperties.fontSize = 20;
// доступ к тексту третьего абзаца
boost::optional<Paragraph> paragraphOpt = document.getBlocks().getParagraph(2);
if (paragraphOpt.has_value()) {
    Range range = paragraphOpt.get().getRange();
    // установить свойства фрагмента текста
    range.setTextProperties(textProperties);
}
```

6.125 Класс TextWrapType

В таблице 65 представлены варианты обтекания текстом встроенного объекта. Используется в `InlineFrame::setWrapType()`.

Таблица 65 – Варианты обтекания текстом встроенного объекта

Константа	Описание
`TextWrapType::Inline`	Встроенный объект располагается в тексте
`TextWrapType::InFrontOfText`	Встроенный объект располагается перед текстом
`TextWrapType::BehindText`	Встроенный объект располагается за текстом
`TextWrapType::TopAndBottom`	Текст располагается сверху и снизу от встроенного объекта
`TextWrapType::Square`	Текст располагается вокруг прямоугольной рамки встроенного объекта
`TextWrapType::Through`	Текст обтекает встроенный объект по сторонам и внутри

6.126 Класс TimePatterns

Форматы времени представлены в таблице 66. Пример использования см. в главе `DateTimeCellFormatting`.

Таблица 66 – Форматы времени

Наименование константы	Описание
`TimePatterns::ShortTime`	`'hh:mm AM/PM'` для языка `en_US`
`TimePatterns::LongTime`	`'hh:mm:ss AM/PM'` для языка `en_US`

6.127 Класс ThemeColorID

В таблице 67 представлены типы идентификаторов цветов тем. Используется в `Color`.

Таблица 67 – Типы идентификаторов цветов тем

Наименование константы	Описание
`ThemeColorID::Background1`	Фон1
`ThemeColorID::Text1`	Текст1
`ThemeColorID::Background2`	Фон2
`ThemeColorID::Text2`	Текст2
`ThemeColorID::Dark1`	Темная1
`ThemeColorID::Dark2`	Темная2

Наименование константы	Описание
`ThemeColorID::Light1`	Светлая1
`ThemeColorID::Light2`	Светлая2
`ThemeColorID::Accent1`	Акцент1
`ThemeColorID::Accent2`	Акцент2
`ThemeColorID::Accent3`	Акцент3
`ThemeColorID::Accent4`	Акцент4
`ThemeColorID::Accent5`	Акцент5
`ThemeColorID::Accent6`	Акцент6
`ThemeColorID::Hyperlink`	Гиперссылка
`ThemeColorID::FollowedHyperlink`	Следующая гиперссылка

6.128 Класс TimeZone

Класс `TimeZone` предоставляет настройки, необходимые для экспорта текстовых документов.

Поле класса `TimeZone.offsetInSecondsToUTC` (числовой тип) содержит значение, с помощью которого задается смещение или разность между временем в данном часовом поясе и временем в формате UTC (Всемирное координированное время).

6.129 Класс TrackedChange

Класс `TrackedChange` представляет отслеживаемое изменение в диапазоне документа (см. Рисунок 40).

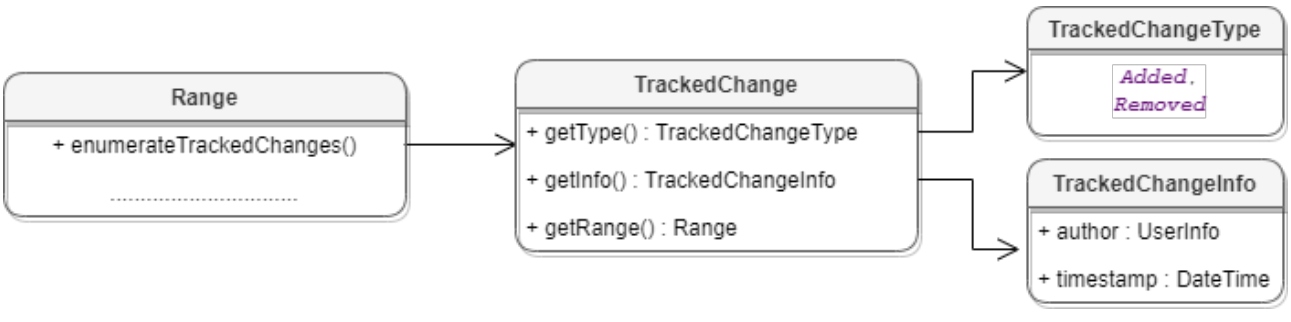


Рисунок 40 – Объектная модель классов для работы с отслеживаемыми изменениями

Для получения списка отслеживаемых изменений используется метод `Range::enumerateTrackedChanges()`.

Пример:

```
std::shared_ptr<Enumerator<TrackedChange>> trackedChangesEnumerator =
document.getRange().getTrackedChangesEnumerator();
if (trackedChangesEnumerator) {
    while (trackedChangesEnumerator->isValid()) {
        TrackedChange trackedChange = trackedChangesEnumerator->getCurrent();
        std::printf("Tracked change range text : %s",
trackedChange.getRange().extractText().c_str());
        trackedChangesEnumerator->goToNext();
    }
}
```

6.129.1 Метод TrackedChange::getRange

Метод возвращает объект Range, который соответствует измененному диапазону внутри абзаца.

Пример:

```
std::shared_ptr<Enumerator<TrackedChange>> trackedChangesEnumerator =
document.getRange().getTrackedChangesEnumerator();
if (trackedChangesEnumerator) {
    while (trackedChangesEnumerator->isValid()) {
        TrackedChange trackedChange = trackedChangesEnumerator->getCurrent();
        std::printf("Tracked change range text : %s",
trackedChange.getRange().extractText().c_str());
        trackedChangesEnumerator->goToNext();
    }
}
```

6.129.2 Метод TrackedChange::getType

Метод позволяет получить информацию о типе отслеживаемого изменения TrackedChangeType.

Пример:

```
std::shared_ptr<Enumerator<TrackedChange>> trackedChangesEnumerator =
document.getRange().getTrackedChangesEnumerator();
if (trackedChangesEnumerator) {
    while (trackedChangesEnumerator->isValid()) {
        TrackedChange trackedChange = trackedChangesEnumerator->getCurrent();
```

```
        TrackedChangeType trackedChangeType = trackedChange.getType();
        bool isAdded = trackedChangeType == TrackedChangeType::Added;
        std::printf("Tracked change type : %s", isAdded ? "Added" : "Removed");
        trackedChangesEnumerator->goToNext();
    }
}
```

6.129.3 Метод TrackedChange::getInfo

Метод позволяет получить информацию об отслеживаемых изменениях TrackedChangeInfo.

Пример:

```
std::shared_ptr<Enumerator<TrackedChange>> trackedChangesEnumerator =
document.getRange().getTrackedChangesEnumerator();
if (trackedChangesEnumerator) {
    while (trackedChangesEnumerator->isValid()) {
        TrackedChange trackedChange = trackedChangesEnumerator->getCurrent();
        TrackedChangeInfo trackedChangeInfo = trackedChange.getInfo();
        ........................................
        trackedChangesEnumerator->goToNext();
     }
}
```

6.130 Класс TrackedChangeInfo

Класс `TrackedChangeInfo` содержит информацию об отслеживаемых изменениях. Описание полей представлено в таблице 68.

Таблица 68 – Описание полей класса `TrackedChangeInfo`

Поле	Тип	Описание
`TrackedChangeInfo.author`	UserInfo	Автор изменений
`TrackedChangeInfo.timeStamp`	DateTime	Дата и время изменений

Пример:

```
std::shared_ptr<Enumerator<TrackedChange>> trackedChangesEnumerator =
document.getRange().getTrackedChangesEnumerator();
if (trackedChangesEnumerator) {
    while (trackedChangesEnumerator->isValid()) {
        TrackedChange trackedChange = trackedChangesEnumerator->getCurrent();
```

```
        TrackedChangeInfo trackedChangeInfo = trackedChange.getInfo();
        if (trackedChangeInfo.author.has_value()) {
            std::printf("Author: %s", trackedChangeInfo.author.get());
        }
        if (trackedChangeInfo.timeStamp.has_value()) {
            std::printf("Year: %d", trackedChangeInfo.timeStamp.get().year);
        }
        trackedChangesEnumerator->goToNext();
    }
}
```

6.130.1 Оператор ==

Оператор сравнения **==** используется для определения эквивалентности значений двух структур TrackedChangeInfo.

```
bool operator == (const TrackedChangeInfo& other) const;
```

6.130.2 Оператор !=

Оператор сравнения **!=** используется для определения неэквивалентности значений двух структур TrackedChangeInfo.

```
bool operator != (const TrackedChangeInfo& other) const;
```

6.131 Класс TrackedChangeType

Класс TrackedChangeType содержит типы отслеживаемых изменений.

```
enum class TrackedChangeType
{
    Added,
    Removed
};
```

Типы отслеживаемых изменений:

- Added – добавленные изменения;
- Removed – удаленные изменения.

Пример:

```
std::shared_ptr<Enumerator<TrackedChange>> trackedChangesEnumerator =
document.getRange().getTrackedChangesEnumerator();
if (trackedChangesEnumerator) {
    while (trackedChangesEnumerator->isValid()) {
```

```
        TrackedChange trackedChange = trackedChangesEnumerator->getCurrent();
        TrackedChangeType type = trackedChange.getType();
        ..................................
        trackedChangesEnumerator->goToNext();
    }
}
```

6.132 Класс UserInfo

Класс `UserInfo` предоставляет информацию о пользователе.

Описание полей класса `UserInfo` представлено в таблице 69.

Таблица 69 – Описание полей класса `UserInfo`

Поле	Описание	Тип
`UserInfo.name`	Имя пользователя	Строка
`UserInfo.email`	Адрес электронной почты пользователя	Строка

6.133 Класс ValueFieldsOrientation

Класс `ValueFueldsOrientation` описывает варианты ориентации в случае, когда в сводной таблице более, чем одно поле из области значений. Является полем класса `PivotTableLayoutSettings`. Описание полей представлено в таблице 70.

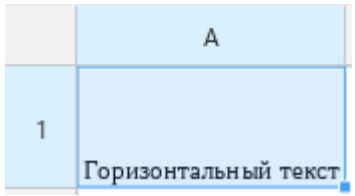
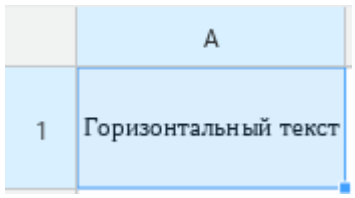
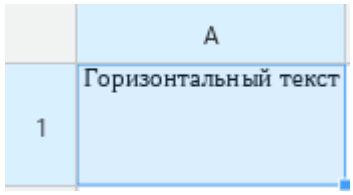
Таблица 70 – Описание полей `ValueFueldsOrientation`

Поле	Описание
`ValueFueldsOrientation::ByRows`	По строкам
`ValueFueldsOrientation::ByColumns`	По столбцам

6.134 Класс VerticalAlignment

В таблице 71 представлены константы,описывающие варианты выравнивания текста по вертикали. Используется в `CellProperties`, `ShapeProperties`.

Таблица 71 – Виды выравнивания текста по вертикали

Наименование константы	Представление в интерфейсе	
`VerticalAlignment::Bottom`		
`VerticalAlignment::Center`		
`VerticalAlignment::Top`		

Пример:

```
Table firstSheet = document.getBlocks().getTable(0).get();
Cell cell = firstSheet.getCell("A3");

CellProperties cellProps = cell.getCellProperties();
cellProps.verticalAlignment = VerticalAlignment::Center;

cell.setCellProperties(cellProps);
```

6.135 Класс VerticalAnchorAlignment

В таблице 72 представлены типы выравнивания объекта относительно закрепленной позиции по вертикали.

Таблица 72 – Типы выравнивания объекта относительно закрепленной позиции по вертикали

Наименование константы	Описание
`VerticalAnchorAlignment::Top`	По верхнему краю
`VerticalAnchorAlignment::Bottom`	По нижнему краю
`VerticalAnchorAlignment::Center`	По центру
`VerticalAnchorAlignment::Inside`, `VerticalAnchorAlignment::Outside`	По границам

6.136 Класс VerticalRelativeTo

В таблице 73 представлены типы размещения объекта относительно закрепленной позиции по вертикали.

Таблица 73 – Типы размещения объекта относительно закрепленной позиции по вертикали

Наименование константы	Описание
`VerticalRelativeTo::Character`	Символ
`VerticalRelativeTo::BaseLine`	Базовая линия
`VerticalRelativeTo::Paragraph`	Абзац
`VerticalRelativeTo::Page`	Страница
`VerticalRelativeTo::PageContent`	Содержимое страницы
`VerticalRelativeTo::PageTopMargin`	Верхнее поле страницы
`VerticalRelativeTo::PageBottomMargin`	Нижнее поле страницы
`VerticalRelativeTo::PageInsideMargin`	Внутреннее поле страницы
`VerticalRelativeTo::PageOutsideMargin`	Внешнее поле страницы

6.137 Класс VerticalTextAnchoredPosition

Класс `VerticalTextAnchoredPosition` предназначен для управления относительным положением объекта со смещением или выравниванием по вертикали.

Объекты могут быть созданы с помощью следующих констукторов:

```
VerticalTextAnchoredPosition(VerticalRelativeTo relativeTo);
VerticalTextAnchoredPosition(VerticalRelativeTo relativeTo, Unit offset);
```

```
VerticalTextAnchoredPosition(VerticalRelativeTo relativeTo,
VerticalAnchorAlignment alignmentType);
```

Описание полей класса `VerticalTextAnchoredPosition` представлено в таблице 74.

Таблица 74 – Описание полей класса `VerticalTextAnchoredPosition`

Поле	Описание
`VerticalTextAnchoredPosition::relativeTo`	Тип размещения объекта относительно закрепленной позиции по вертикали `VerticalRelativeTo`
`VerticalTextAnchoredPosition::offset`	Смещение объекта
`VerticalTextAnchoredPosition::aligment`	Тип выравнивания объекта относительно закрепленной позиции по вертикали `VerticalAnchorAlignment`

Примеры:

```
// Использование конструкторов
vertTextAnchoredPos =
VerticalTextAnchoredPosition(VerticalRelativeTo::Character);
vertTextAnchoredPos =
VerticalTextAnchoredPosition(VerticalRelativeTo::Character, 10.0);
vertTextAnchoredPos =
VerticalTextAnchoredPosition(VerticalRelativeTo::Character,
VerticalAnchorAlignment::Inside);

// Непосредственное обращение к полям
vertTextAnchoredPos.offset = 15.0;
vertTextAnchoredPos.relativeTo = VerticalRelativeTo::PageBottomMargin;
vertTextAnchoredPos.alignment = VerticalAnchorAlignment::Inside;
```

6.137.1 Оператор ==

Метод используется для определения эквивалентности двух объектов `VerticalTextAnchoredPosition`.

6.137.2 Оператор !=

Метод используется для определения неэквивалентности двух объектов `VerticalTextAnchoredPosition`.

6.138 Класс WorkbookExportSettings

Класс `WorkbookExportSettings` предоставляет настройки, необходимые для экспорта табличных документов (см. `document::exportAs()`).

Описание полей класса `WorkbookExportSettings` представлено в таблице 75.

Таблица 75 – Описание полей класса `WorkbookExportSettings`

Поле	Описание
`WorkbookExportSettings.sheetNames`	Представляет коллекцию имен листов для экспорта, тип `std::vector<std::string>`. Если коллекция пуста, экспортируются все листы.
`WorkbookExportSettings.printingScope`	Представляет область печати (весь документ, область печати, пользовательский диапазон и т. д.) `PrintingScope`.
`WorkbookExportSettings.pageProperties`	Представляют свойства страницы для выходного документа (высота и ширина страницы в пунктах pt) `PageProperties`.
`WorkbookExportSettings.scale`	Представляет масштаб экспорта выходного документа в процентах (например, 50,0%, 150,63%, 400,0% и т. д.).

Пример:

```
WorkbookExportSettings workbookSettings = WorkbookExportSettings();
workbookSettings.sheetNames = std::vector<std::string>();
workbookSettings.sheetNames.push_back("Лист2");
workbookSettings.printingScope = PrintingScope(PrintingScope::Type::PrintArea);
workbookSettings.pageProperties = PageProperties(100, 200);
workbookSettings.scale = 90;
document.exportAs(filePath, ExportFormat::PDFA1, workbookSettings);
```

6.139 Исключения

6.139.1 Класс BaseError

Класс `BaseError` является базовым классом для всех исключений SDK.

```
class BaseError : public std::runtime_error
{
public:
```

```
    explicit BaseError(const std::string& error);
};
```

6.139.2 Класс ApplicationCreateError

Исключение `ApplicationCreateError` вызывается в случае, когда объект `Application` не может быть создан.

```
class ApplicationCreateError : public BaseError
{
public:
    explicit ApplicationCreateError(const std::string& error);
};
```

6.139.3 Класс IncorrectArgumentError

Исключение `IncorrectArgumentError` вызывается в случае, когда один из аргументов метода или функции имеет недействительное значение.

```
class IncorrectArgumentError : public BaseError
{
public:
    IncorrectArgumentError(const std::string& argumentName,
                           const std::string& argumentValue,
                           const std::string& reason);
};
```

6.139.4 Класс InvalidObjectError

Исключение `InvalidObjectError` вызывается в случае, когда объект больше не может быть использован.

```
class InvalidObjectError : public BaseError
{
public:
    InvalidObjectError();
    InvalidObjectError(const std::string& objectName);
};
```

6.139.5 Класс DocumentCreateError

Исключение `DocumentCreateError` вызывается в случае, когда документ не может быть создан.

```
class DocumentCreateError : public BaseError
{
```

```
public:
    explicit DocumentCreateError(const std::string& error);
};
```

6.139.6 Класс DocumentLoadError

Исключение `DocumentLoadError` вызывается в случае, когда документ не может быть загружен.

```
class DocumentLoadError : public BaseError
{
public:
    explicit DocumentLoadError(const std::string& error);
};
```

6.139.7 Класс DocumentSaveError

Исключение `DocumentSaveError` вызывается в случае, когда документ не может быть сохранен.

```
class DocumentSaveError : public BaseError
{
public:
    explicit DocumentSaveError(const std::string& error);
};
```

6.139.8 Класс DocumentExportError

Исключение `DocumentExportError` вызывается в случае, когда документ не может быть экспортирован.

```
class DocumentExportError : public BaseError
{
public:
    explicit DocumentExportError(const std::string& error);
};
```

6.139.9 Класс NoSuchElementError

Исключение `NoSuchElementError` вызывается в случае, когда элемент не существует.

```
class NoSuchElementError : public BaseError
{
public:
```

```
    NoSuchElementError();
};
```

6.139.10 Класс NotImplementedError

Исключение `NotImplementedError` вызывается в случае, если обнаружена нереализованная функциональность.

```
class NotImplementedError : public BaseError
{
public:
    explicit NotImplementedError(const std::string& error);
};
```

6.139.11 Класс OutOfRangeError

Исключение `OutOfRangeError` вызывается в случае обнаружения выхода значения за пределы диапазона.

```
class OutOfRangeError : public BaseError
{
public:
    explicit OutOfRangeError(const std::string& rangeAsStr);
};
```

6.139.12 Класс ParseError

Исключение `ParseError` вызывается в случае, когда текст не прошел синтаксический анализ.

```
class ParseError : public BaseError
{
public:
    explicit ParseError(const std::string& error);
};
```

6.139.13 Класс UnknownError

Исключение `UnknownError` вызывается в случае, когда критическое исключение возникло по неизвестной причине. Приложение должно быть завершено, поскольку возникло неопределенное состояние ядра Document API.

```
class UnknownError : public BaseError
{
public:
```

```
    UnknownError();
    explicit UnknownError(const std::string& error);
};
```

6.139.14 Класс ForbiddenActionError

Исключение `ForbiddenActionError` вызывается в случае выполнения запрещенной операции.

```
class ForbiddenActionError : public BaseError
{
public:
    explicit ForbiddenActionError(const std::string& reason);
};
```

6.139.15 Класс DocumentModificationError

Исключение `DocumentModificationError` вызывается в случае, когда невозможно выполнить операцию по изменению документа.

```
class DocumentModificationError : public BaseError
{
    public:
    DocumentModificationError(const std::string& reason);
};
```

6.139.16 Класс PivotTableError

Исключение `PivotTableError` вызывается в случае ошибки при работе со сводными таблицами. Например, использование фильтра, который не может быть применен к сводной таблице.

```
class PivotTableError : public BaseError
{
    public:
    explicit PivotTableError(const std::string& error);
};
```

6.139.17 Класс PositionDocumentsMismatchError

Исключение `PositionDocumentsMismatchError` вызывается в случае, когда несколько позиций относятся к различным документам и не могут быть использованы в одной операции.

Например, при попытке пользователя создать диапазон (`Range`), включающий позиции (`Position`), принадлежащие нескольким различным документам, и выполнить операцию для такого диапазона.

```
class PositionDocumentsMismatchError : public BaseError
{
    public:
    PositionDocumentsMismatchError();
};
```

6.139.18 Класс ScriptExecutionError

Исключение `ScriptExecutionError` вызывается в случае, когда сценарий не удается выполнить.

```
try {
    scripting->runScript("ScriptName");
}
catch (const ScriptExecutionError& error) {
    std::printf("%s", error.what());
}
```