



МойОфис
**Комплект Средств
Разработки (SDK)**

Руководство программиста

MYOFFICE DOCUMENT API (C++)

2022.01

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 170 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

1. ОБЩИЕ СВЕДЕНИЯ	23
1.1 Назначение программы	23
1.2 Библиотека MyOffice Document API для языка программирования C++	23
1.3 Уровень подготовки пользователя	24
1.4 Системные требования	24
2. ПОДГОТОВКА К РАБОТЕ	25
2.1 Список дистрибутивов	25
2.2 Установка	25
2.3 Сборка приложения для ОС Microsoft Windows	26
2.3.1 Сборка приложения с использованием IDE	27
2.3.1.1 Настройка и сборка приложения в среде Microsoft Visual Studio	27
2.3.1.2 Проверка работоспособности	30
2.3.2 Сборка приложения из командной строки	31
2.3.2.1 Сценарий сборки	31
2.3.2.2 Сборка приложения	32
2.3.2.3 Проверка работоспособности	32
2.3.3 Распространение разработанных приложений	33
2.4 Сборка приложения для ОС Linux	33
2.4.1 Сценарий сборки	34
2.4.2 Сборка приложения	35
2.4.3 Проверка работоспособности	35
2.4.4 Распространение разработанных приложений	35
3. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ	36
3.1 Проверка версии библиотеки	36
3.2 Работа с текстовым документом	36
3.2.1 Создание и сохранение документа	36
3.2.2 Работа с текстом	37
3.2.2.1 Настройка свойств текста	37
3.2.3 Работа с таблицами	37
3.2.3.1 Вставка таблицы	37
3.2.3.2 Удаление таблицы	38
3.2.4 Работа с ячейками таблицы	38

3.2.4.1	Объединение ячеек таблицы	38
3.2.4.2	Разъединение ячеек таблицы	38
3.2.4.3	Поворот текста в ячейке	39
3.2.5	Форматирование таблицы	39
3.2.5.1	Установка свойств форматирования ячеек таблицы	39
3.2.5.2	Установка границ ячеек таблицы	39
3.2.6	Работа с закладками	40
3.2.6.1	Вставка закладки	40
3.2.6.2	Изменение содержимого закладки	40
3.2.6.3	Удаление закладки	41
3.2.7	Работа с комментариями	41
3.2.7.1	Получение списка комментариев	41
3.2.7.2	Получение списка ответов	41
3.2.8	Экспорт текстового документа	42
3.2.9	Поиск в текстовом документе	42
3.2.10	Управление ориентацией и свойствами страниц раздела	43
3.2.11	Работа с отслеживаемыми изменениями	43
3.2.12	Работа с колонтитулами раздела	44
3.3	Работа с табличным документом	44
3.3.1	Создание и сохранение документа	44
3.3.2	Работа с текстом	45
3.3.2.1	Настройка свойств текста	45
3.3.3	Работа с листами табличного документа	45
3.3.3.1	Вставка рабочего листа в табличный документ	45
3.3.3.2	Удаление рабочего листа табличного документа	45
3.3.4	Работа с ячейками таблицы	46
3.3.4.1	Настройка свойств ячейки	46
3.3.4.2	Объединение ячеек таблицы	47
3.3.4.3	Разъединение ячеек таблицы	47
3.3.5	Экспорт табличного документа	47
3.3.6	Поиск в табличном документе	48
3.3.7	Работа со сводными таблицами	49
3.3.7.1	Получение диапазона исходных данных сводной таблицы	49
3.3.7.2	Получение диапазона размещения сводной таблицы	49

3.3.7.3	Получение неподдерживаемых свойств сводной таблицы	49
3.3.7.4	Получение флагов отображения общих итогов для строк и колонок	49
3.3.7.5	Получение заголовков сводной таблицы	49
3.3.7.6	Получение и применение фильтра для сводной таблицы	49
3.3.7.7	Получение полей из области фильтров	50
3.3.7.8	Получение полей из области значений	50
3.3.7.9	Получение полей из области строк	50
3.3.7.10	Получение полей из области колонок	50
3.3.7.11	Получение настроек отображения сводной таблицы	51
3.3.7.12	Обновление сводной таблицы	51
3.3.8	Работа с именованными выражениями	51
3.3.8.1	Получение именованного выражения	51
3.3.8.2	Получение именованного выражения таблицы	51
3.3.8.3	Получение свойств именованного выражения	51
3.3.8.4	Получение коллекции именованных выражений	51
3.4	Работа со встроенными объектами	52
4.	СПРАВОЧНИК КЛАССОВ, СТРУКТУР И МЕТОДОВ	54
4.1	Типы документов	54
4.1.1	Класс DocumentType	54
4.2	Форматы документов	55
4.2.1	Класс DocumentFormat	55
4.3	Форматы экспорта документов	56
4.3.1	Класс ExportFormat	56
4.4	Неподдерживаемые свойства документа	56
4.4.1	Структура SaveUnsupportedFeatures	56
4.5	Системы адресации ячеек	57
4.5.1	Класс FormulaType	57
4.6	Кодировки документов	57
4.6.1	Класс Encoding	57
4.7	Типы выравнивания текста	58
4.7.1	Класс Alignment	58
4.7.2	Класс TextLayout	58
4.7.3	Класс VerticalAlignment	59
4.7.4	Класс TextWrapType	60

4.8	Стили линий	61
4.8.1	Класс LineStyle	61
4.9	Типы надстрочного и подстрочного форматирования	61
4.9.1	Класс ScriptPosition	61
4.10	Типы форматов ячеек	62
4.10.1	Класс CellFormat	62
4.10.2	Структура AccountingCellFormatting	62
4.10.3	Структура PercentageCellFormatting	63
4.10.4	Структура NumberCellFormatting	63
4.10.5	Структура CurrencyCellFormatting	64
4.10.6	Структура DateTimeCellFormatting	64
4.10.6.1	Класс DatePatterns	65
4.10.6.2	Класс TimePatterns	65
4.10.7	Структура FractionCellFormatting	66
4.10.8	Структура ScientificCellFormatting	66
4.11	Типы межстрочного интервала	66
4.11.1	Класс LineSpacingRule	66
4.12	Типы схем форматирования списков	67
4.12.1	Класс ListSchema	67
4.13	Типы отслеживаемых изменений	69
4.13.1	Класс TrackedChangeType	69
4.14	Типы колонтитулов	69
4.14.1	Класс HeaderFooterType	69
4.15	Масштабирование при печати табличных документов	70
4.15.1	Класс WorksheetPrinterFitType	70
4.16	Выбор страниц для экспорта и печати	70
4.16.1	Класс PageParity	70
4.17	Типы переменных	71
4.17.1	Тип Percents	71
4.17.2	Тип SheetNames	71
4.17.3	Тип HorizontalTextAnchoredPosition	71
4.17.4	Тип VerticalTextAnchoredPosition	71
4.17.5	Тип HorizontalTypeTraits	71
4.17.6	Тип VerticalTypeTraits	71

4.18	Типы идентификаторов цветов тем	72
4.18.1	Класс ThemeColorID	72
4.19	Типы окончаний линий	73
4.19.1	Класс LineEndingStyle	73
4.20	Типы размещения объекта по горизонтали	73
4.20.1	Класс HorizontalRelativeTo	73
4.21	Типы размещения объекта по вертикали	74
4.21.1	Класс VerticalRelativeTo	74
4.22	Типы выравнивания объекта по вертикали	75
4.22.1	Класс VerticalAnchorAlignment	75
4.23	Типы выравнивания объекта по горизонтали	75
4.23.1	Класс HorizontalAnchorAlignment	75
4.24	Классы и структуры, относящиеся к ядру (Core Types)	76
4.24.1	Класс Application	76
4.24.1.1	Метод Application	76
4.24.1.2	Метод getMessenger	76
4.24.1.3	Метод createDocument	76
4.24.1.4	Метод loadDocument	77
4.24.2	Класс Messenger	77
4.24.2.1	Метод subscribe	77
4.24.2.2	Метод notify	77
4.24.3	Класс Connection	77
4.24.4	Класс Message	78
4.24.4.1	Оператор ==	78
4.24.4.2	Оператор !=	78
4.24.4.3	Метод getSeverity	78
4.24.4.4	Метод getText	78
4.24.4.5	Метод makeInfo	79
4.24.4.6	Метод makeWarning	79
4.24.4.7	Метод makeError	79
4.24.5	Класс MessageHandler	79
4.24.6	Структура DocumentSettings	79
4.24.7	Структура LoadDocumentSettings	80
4.24.8	Структура SaveDocumentSettings	80

4.24.9	Структура UserInfo	81
4.24.10	Класс CurrencySignPlacement	81
4.24.11	Структура LocaleInfo	81
4.24.12	Структура TimeZone	82
4.24.13	Структура DSVSettings	82
4.24.14	Структура Point	83
4.24.14.1	Метод Point	83
4.24.14.2	Метод toString	83
4.24.15	Структура Size	83
4.24.15.1	Метод Size	84
4.24.15.2	Метод toString	84
4.24.16	Структура Rect	84
4.24.16.1	Метод Rect	84
4.24.16.2	Метод toString	85
4.24.17	Структура ColorRGBA	85
4.24.17.1	Метод ColorRGBA	85
4.24.17.2	Оператор ==	85
4.24.17.3	Оператор !=	85
4.24.17.4	Поля структуры ColorRGBA	86
4.24.18	Структура DateTime	86
4.24.18.1	Поля структуры DateTime	86
4.24.18.2	Оператор ==	86
4.24.18.3	Оператор !=	86
4.25	Классы, структуры и методы объектной модели документа	87
4.25.1	Класс Document	87
4.25.1.1	Метод saveAs	87
4.25.1.2	Метод exportAs	88
4.25.1.3	Метод validate	88
4.25.1.4	Метод getBlocks	88
4.25.1.5	Метод getBookmarks	88
4.25.1.6	Метод getNamedExpressions	89
4.25.1.7	Метод getScripts	89
4.25.1.8	Метод getRange	89
4.25.1.9	Метод getComments	89

4.25.1.10	Метод setChangesTrackingEnabled	89
4.25.1.11	Метод isChangesTrackingEnabled	89
4.25.1.12	Метод merge	89
4.25.1.13	Метод setPageProperties	89
4.25.1.14	Метод setPageOrientation	89
4.25.1.15	Метод getSections	90
4.25.1.16	Метод setMirroredMarginsEnabled	90
4.25.1.17	Метод areMirroredMarginsEnabled	90
4.25.1.18	Метод getPivotTablesManager	90
4.25.2	Класс Table	90
4.25.2.1	Наименование таблицы	91
4.25.2.2	Размеры таблицы	92
4.25.2.3	Управление столбцами и строками	92
4.25.2.4	Управление ячейками и диапазонами ячеек	94
4.25.2.5	Список диаграмм	94
4.25.2.6	Список именованных выражений	94
4.25.2.7	Создание копии листа в табличном документе	94
4.25.2.8	Перемещение листа в табличном документе	95
4.25.2.9	Управление видимостью листа в табличном документе	95
4.25.2.10	Область печати в табличном документе	95
4.25.2.11	Оператор ==	95
4.25.2.12	Оператор !=	95
4.25.3	Именованные выражения	96
4.25.3.1	Класс NamedExpression	96
4.25.3.1.1	Метод getName	96
4.25.3.1.2	Метод getExpression	96
4.25.3.1.3	Метод getCellRange	96
4.25.3.2	Класс NamedExpressions	96
4.25.3.2.1	Метод get	97
4.25.3.2.2	Метод getEnumerator	97
4.25.3.3	Класс NamedExpressionsValidationResult	97
4.25.4	Сводные таблицы	97
4.25.4.1	Класс PivotTable	97
4.25.4.1.1	Метод remove	98

4.25.4.1.2	Метод getSourceRangeAddress	98
4.25.4.1.3	Метод getSourceRange	98
4.25.4.1.4	Метод getPivotRange	98
4.25.4.1.5	Метод changeSourceRange	99
4.25.4.1.6	Метод isRowGrandTotalEnabled	99
4.25.4.1.7	Метод isColumnGrandTotalEnabled	99
4.25.4.1.8	Метод getPivotTableCaptions	99
4.25.4.1.9	Метод getPivotTableLayoutSettings	99
4.25.4.1.10	Метод getUnsupportedFeatures	99
4.25.4.1.11	Метод getFieldsList	99
4.25.4.1.12	Метод getRowFields	99
4.25.4.1.13	Метод getColumnFields	100
4.25.4.1.14	Метод getValueFields	100
4.25.4.1.15	Метод getPageFields	100
4.25.4.1.16	Метод getFieldCategories	100
4.25.4.1.17	Метод getFieldItems	100
4.25.4.1.18	Метод getFieldItemsByName	100
4.25.4.1.19	Метод getFilter	100
4.25.4.1.20	Метод getFilters	100
4.25.4.1.21	Метод update	100
4.25.4.1.22	Метод createPivotTableEditor	100
4.25.4.2	Структура PivotTableCaptions	101
4.25.4.3	Структура PivotTableLayoutSettings	101
4.25.4.4	Класс PivotTableUnsupportedFeature	102
4.25.4.5	Класс PivotTableReportLayout	103
4.25.4.6	Класс ValueFieldsOrientation	103
4.25.4.7	Класс PageFieldOrder	103
4.25.4.8	Класс PivotTableFieldCategory	104
4.25.4.9	Класс PivotTableFieldCategories	104
4.25.4.9.1	Метод getEnumerator	104
4.25.4.10	Класс PivotTableFunction	104
4.25.4.11	Класс PivotTableFilter	105
4.25.4.11.1	Метод getFieldName	106
4.25.4.11.2	Метод getCount	106

4.25.4.11.3	Метод getName	106
4.25.4.11.4	Метод isHidden	106
4.25.4.11.5	Метод setHidden	106
4.25.4.12	Класс PivotTableFilters	106
4.25.4.12.1	Метод GetEnumerator	106
4.25.4.13	Структура PivotTableFieldProperties	106
4.25.4.14	Структура PivotTableField	107
4.25.4.15	Структура PivotTableCategoryField	107
4.25.4.16	Структура PivotTableValueField	107
4.25.4.17	Структура PivotTablePageField	108
4.25.4.18	Класс PivotTableItem	108
4.25.4.18.1	Метод getName	108
4.25.4.18.2	Метод getAlias	108
4.25.4.18.3	Метод getItemType	109
4.25.4.18.4	Метод isCollapsed	109
4.25.4.19	Класс PivotTableItemType	109
4.25.4.20	Класс PivotTableEditor	109
4.25.4.20.1	Метод addField	110
4.25.4.20.2	Метод moveField	110
4.25.4.20.3	Метод removeField	111
4.25.4.20.4	Метод reorderField	111
4.25.4.20.5	Метод enableField	111
4.25.4.20.6	Метод disableField	111
4.25.4.20.7	Метод setSummmarizeFunction	111
4.25.4.20.8	Метод setFilter	111
4.25.4.20.9	Метод setFilters	111
4.25.4.20.10	Метод setCaptions	112
4.25.4.20.11	Метод setLayoutSettings	112
4.25.4.20.12	Метод setGrandTotalSettings	112
4.25.4.20.13	Метод apply	112
4.25.4.21	Класс PivotTableUpdateResult	112
4.25.4.22	Класс PivotTablesManager	113
4.25.4.22.1	Метод create	113
4.25.5	Диаграммы	114

4.25.5.1	Класс Chart	114
4.25.5.1.1	Метод getType	114
4.25.5.1.2	Метод setType	114
4.25.5.1.3	Метод getRangesCount	115
4.25.5.1.4	Метод getRange	115
4.25.5.1.5	Метод getTitle	115
4.25.5.1.6	Метод setRange	115
4.25.5.1.7	Метод setRect	115
4.25.5.1.8	Метод isEmpty	115
4.25.5.1.9	Метод isSolidRange	115
4.25.5.1.10	Метод is3D	115
4.25.5.1.11	Метод getDirectionType	116
4.25.5.1.12	Метод getChartLabels	116
4.25.5.1.13	Метод getRangeAsString	116
4.25.5.1.14	Метод applySettings	116
4.25.5.2	Класс Charts	116
4.25.5.2.1	Метод getChartsCount	116
4.25.5.2.2	Метод getChart	117
4.25.5.2.3	Метод getChartIndexByDrawingIndex	117
4.25.5.3	Класс ChartLabelsDetectionMode	117
4.25.5.4	Структура ChartLabelsInfo	117
4.25.5.5	Структура ChartRangeInfo	118
4.25.5.6	Класс ChartRangeType	118
4.25.5.7	Класс ChartSeriesDirectionType	119
4.25.5.8	Класс ChartType	119
4.25.6	Структура TableRangeInfo	120
4.25.6.1	Конструктор TableRangeInfo	120
4.25.6.2	Поля структуры TableRangeInfo	120
4.25.7	Класс Paragraphs	121
4.25.7.1	Метод setListSchema	121
4.25.7.2	Методы setListLevel, increaseListLevel, decreaseListLevel	121
4.25.7.3	Метод getEnumerator	121
4.25.8	Класс Paragraph	122
4.25.8.1	Методы getParagraphProperties, setParagraphProperties	122

4.25.8.2	Методы getListSchema, setListSchema	122
4.25.8.3	Методы getListLevel, setListLevel, increaseListLevel, decreaseListLevel	123
4.25.9	Структура ParagraphProperties	123
4.25.9.1	Оператор ==	124
4.25.9.2	Оператор !=	124
4.25.10	Класс Shape	124
4.25.11	Класс ShapeProperties	124
4.25.12	Класс ShapeTextLayout	124
4.25.13	Класс Field	125
4.25.14	Класс Scripts	125
4.25.14.1	Метод getScript	125
4.25.14.2	Метод setScript	125
4.25.14.3	Метод removeScript	125
4.25.14.4	Метод getEnumerator	126
4.25.15	Класс Script	126
4.25.15.1	Метод getName	126
4.25.15.2	Метод setName	126
4.25.15.3	Метод getBody	126
4.25.15.4	Метод setBody	126
4.25.16	Класс Blocks	126
4.25.16.1	Методы getBlock, getParagraph, getShape, getTable, getField	127
4.25.16.2	Методы getEnumerator, getParagraphsEnumerator, getShapesEnumerator, getTablesEnumerator, getFieldsEnumerator	127
4.25.17	Класс Block	127
4.25.17.1	Методы toParagraph, toTable, toShape, toField	127
4.25.17.2	Метод getRange	128
4.25.17.3	Метод remove	128
4.25.17.4	Метод getSection	128
4.25.18	Класс Borders	128
4.25.18.1	Методы для считывания свойств	129
4.25.18.2	Методы для установки свойств	129
4.25.19	Класс RangeBorders	129
4.25.20	Структура CellPosition	130
4.25.20.1	Метод CellPosition	130

4.25.20.2	Метод toString	130
4.25.20.3	Поля структуры CellPosition	130
4.25.21	Структура CellRangePosition, коллекция CellRangePositions	130
4.25.21.1	Метод CellRangePosition	131
4.25.21.2	Метод toString	131
4.25.21.3	Поля структуры CellRangePosition	131
4.25.22	Класс CellRange	132
4.25.22.1	Метод getTable	132
4.25.22.2	Метод getBeginRow	132
4.25.22.3	Метод getBeginColumn	132
4.25.22.4	Метод getLastRow	132
4.25.22.5	Метод getLastColumn	132
4.25.22.6	Метод getEnumerator	132
4.25.22.7	Метод setBorders	133
4.25.22.8	Метод setCellProperties	133
4.25.22.9	Метод getCellProperties	133
4.25.22.10	Метод autoFill	133
4.25.22.11	Метод merge	133
4.25.23	Класс Cell	133
4.25.23.1	Метод getRange	134
4.25.23.2	Метод getFormat	135
4.25.23.3	Метод setFormat	135
4.25.23.4	Метод getCustomFormat	135
4.25.23.5	Метод setCustomFormat	135
4.25.23.6	Методы setBool, setNumber, setText	135
4.25.23.7	Метод setFormula	135
4.25.23.8	Метод getFormulaAsString	136
4.25.23.9	Метод getFormattedValue	136
4.25.23.10	Метод setFormattedValue	136
4.25.23.11	Метод getRawValue	136
4.25.23.12	Метод setContent	137
4.25.23.13	Методы getCellProperties, setCellProperties	137
4.25.23.14	Метод getBorders	137
4.25.23.15	Метод setBorders	137

4.25.23.16	Метод isPivotTableRoot	137
4.25.23.17	Метод getPivotTable	137
4.25.23.18	Методы getParagraphProperties, setParagraphProperties	137
4.25.23.19	Метод unmerge	138
4.25.24	Структура CellProperties	138
4.25.24.1	Оператор ==	138
4.25.24.2	Оператор !=	138
4.25.25	Структура LineEndingProperties	139
4.25.25.1	Оператор ==	139
4.25.25.2	Оператор !=	139
4.25.26	Структура LineProperties	139
4.25.26.1	Оператор ==	140
4.25.26.2	Оператор !=	140
4.25.27	Структура LineSpacing	140
4.25.27.1	Метод LineSpacing	140
4.25.27.2	Оператор ==	140
4.25.27.3	Оператор !=	140
4.25.27.4	Поля структуры LineSpacing	141
4.25.28	Класс Position	141
4.25.28.1	Метод insertText	141
4.25.28.2	Метод insertTable	141
4.25.28.3	Метод insertPageBreak	142
4.25.28.4	Метод insertLineBreak	142
4.25.28.5	Метод insertBookmark	142
4.25.28.6	Метод insertImage	142
4.25.28.7	Метод operator==	142
4.25.28.8	Метод operator!=	142
4.25.28.9	Метод insertSectionBreak	142
4.25.28.10	Метод removeBackward	142
4.25.28.11	Метод removeForward	143
4.25.29	Класс Range	143
4.25.29.1	Метод Range	143
4.25.29.2	Метод getBegin	143
4.25.29.3	Метод getEnd	144

4.25.29.4	Метод extractText	144
4.25.29.5	Метод removeContent	144
4.25.29.6	Метод lockContent	144
4.25.29.7	Метод unlockContent	144
4.25.29.8	Метод isContentLocked	144
4.25.29.9	Метод replaceText	144
4.25.29.10	Метод getTextProperties	144
4.25.29.11	Метод setTextProperties	145
4.25.29.12	Метод getBlocksEnumerator	145
4.25.29.13	Метод getTrackedChangesEnumerator	145
4.25.29.14	Метод getComments	145
4.25.29.15	Метод getParagraphs	145
4.25.29.16	Метод getImages	145
4.25.29.17	Метод getInlineObjects	145
4.25.30	Класс Search	145
4.25.30.1	Метод findText	146
4.25.30.2	Глобальная функция createSearch	146
4.25.31	Структура TextProperties	146
4.25.32	Класс Bookmarks	147
4.25.32.1	Метод getBookmarkRange	147
4.25.32.2	Метод removeBookmark	147
4.25.33	Класс Comment	147
4.25.33.1	Метод getRange	148
4.25.33.2	Метод isResolved	148
4.25.33.3	Метод getText	148
4.25.33.4	Метод getAudioUrl	148
4.25.33.5	Метод getReplies	148
4.25.33.6	Метод getInfo	148
4.25.34	Класс Comments	148
4.25.34.1	Метод getEnumerator	148
4.25.35	Класс TrackedChange	149
4.25.35.1	Метод TrackedChange	149
4.25.35.2	Метод getRange	149
4.25.35.3	Методы getType	149

4.25.35.4	Методы getInfo	149
4.25.36	Структура TrackedChangeInfo	149
4.25.36.1	Поля структуры TrackedChangeInfo	150
4.25.36.2	Оператор ==	150
4.25.36.3	Оператор !=	150
4.25.37	Класс Section	150
4.25.37.1	Метод setPageProperties	150
4.25.37.2	Метод getPageProperties	150
4.25.37.3	Метод setPageOrientation	151
4.25.37.4	Метод getPageOrientation	151
4.25.37.5	Метод getRange	151
4.25.37.6	Метод getHeaders	151
4.25.37.7	Метод getFooters	151
4.25.38	Класс Sections	151
4.25.38.1	Метод getEnumerator	151
4.25.39	Класс SectionBreakType	151
4.25.40	Класс PageOrientation	152
4.25.41	Структура PageProperties	152
4.25.41.1	Оператор ==	152
4.25.41.2	Оператор !=	152
4.25.42	Структура Insets	153
4.25.42.1	Оператор ==	153
4.25.42.2	Оператор !=	153
4.25.43	Класс HeaderFooter	153
4.25.43.1	Метод getType	153
4.25.43.2	Метод getBlocks	153
4.25.43.3	Метод getRange	154
4.25.44	Класс HeadersFooters	154
4.25.44.1	Метод getEnumerator	154
4.25.45	Класс TextOrientation	154
4.25.45.1	Метод TextOrientation	154
4.25.45.2	Метод getAngle	155
4.25.45.3	Метод isStackedChars	155
4.25.45.4	Оператор ==	155

4.25.45.5	Оператор !=	155
4.25.46	Структура TextExportSettings	155
4.25.47	Структура WorkbookExportSettings	155
4.25.48	Класс PrintingScope	156
4.25.48.1	Класс PrintingScope::Type	156
4.25.48.2	Метод PrintingScope	157
4.25.48.3	Метод usePrintArea	157
4.25.48.4	Метод getCellRange	157
4.25.49	Класс PageNumbers	157
4.25.49.1	Метод PageNumbers	157
4.25.49.2	Метод contains	158
4.25.49.3	Метод getLast	158
4.25.50	Класс Color	158
4.25.50.1	Метод Color	158
4.25.50.2	Метод getRGBAColor	159
4.25.50.3	Метод getThemeColorID	159
4.25.50.4	Метод setTransforms	159
4.25.50.5	Метод getTransforms	159
4.25.50.6	Оператор ==	159
4.25.50.7	Оператор !=	159
4.25.51	Класс ColorTransforms	159
4.25.52	Класс Frame	160
4.25.52.1	Метод setPosition	160
4.25.52.2	Метод getPosition	160
4.25.52.3	Метод getDimensions	160
4.25.52.4	Метод setDimensions	160
4.25.52.5	Метод setWrapType	160
4.25.52.6	Метод getWrapType	161
4.25.53	Класс Image	161
4.25.53.1	Метод getFrame	161
4.25.54	Класс Images	161
4.25.54.1	Метод getEnumerator	161
4.25.55	Структура RelativeAnchoredPosition	161
4.25.55.1	Метод RelativeAnchoredPosition	162

4.25.55.2	Оператор ==	162
4.25.55.3	Оператор !=	162
4.25.56	Структура TextAnchoredPosition	162
4.25.56.1	Метод TextAnchoredPosition	163
4.25.56.2	Оператор ==	163
4.25.56.3	Оператор !=	163
4.25.57	Структура AnchoredPosition	163
4.25.57.1	Метод AnchoredPosition	164
4.25.57.2	Оператор ==	164
4.25.57.3	Оператор !=	164
4.25.58	Класс InlineObject	164
4.25.58.1	Метод toImage	164
4.25.58.2	Метод getFrame	165
4.25.59	Класс InlineObjects	165
4.25.59.1	Метод getEnumerator	165
4.26	Классы и методы для работы с макрокомандами (Scripting)	165
4.26.1	Класс Scripting	165
4.26.1.1	Метод runScript	165
4.26.1.2	Глобальная функция createScripting	165
4.27	Исключения	165
4.27.1	Класс BaseError	166
4.27.2	Класс ApplicationCreateError	166
4.27.3	Класс IncorrectArgumentError	166
4.27.4	Класс InvalidObjectError	166
4.27.5	Класс DocumentCreateError	166
4.27.6	Класс DocumentLoadError	167
4.27.7	Класс DocumentSaveError	167
4.27.8	Класс DocumentExportError	167
4.27.9	Класс NoSuchElementError	167
4.27.10	Класс NotImplementedError	167
4.27.11	Класс OutOfRangeError	168
4.27.12	Класс ParseError	168
4.27.13	Класс UnknownError	168
4.27.14	Класс ForbiddenActionError	168

МойОфис

4.27.15 Класс DocumentModificationError	169
4.27.16 Класс PivotTableError	169
4.27.17 Класс PositionDocumentsMismatchError	169
4.27.18 Класс ScriptExecutionError	169
5. ВЕРСИИ DOCUMENT API	170
5.1 Механизм контроля версий	170
5.2 Изменения	170

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

В настоящем документе используются следующие сокращения (см. [Таблица 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 программы

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/A-1.
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. [Таблица 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. [Таблица 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 поддержки 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.

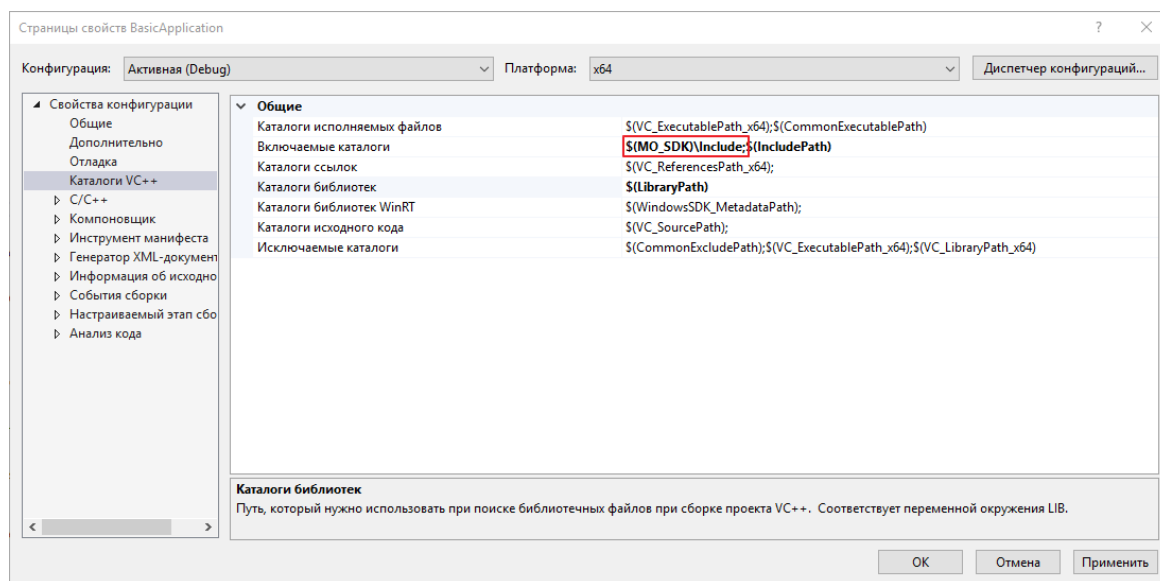


Рисунок 1 – Настройка каталогов включаемых файлов

2. Указать каталог для поиска файлов библиотек, используя переменную окружения **MO_SDK** (см. [Рисунок 2](#)). Файлы библиотек расположены в папке **MyOfficeDocumentAPIlib** каталога установки MyOffice Document API.

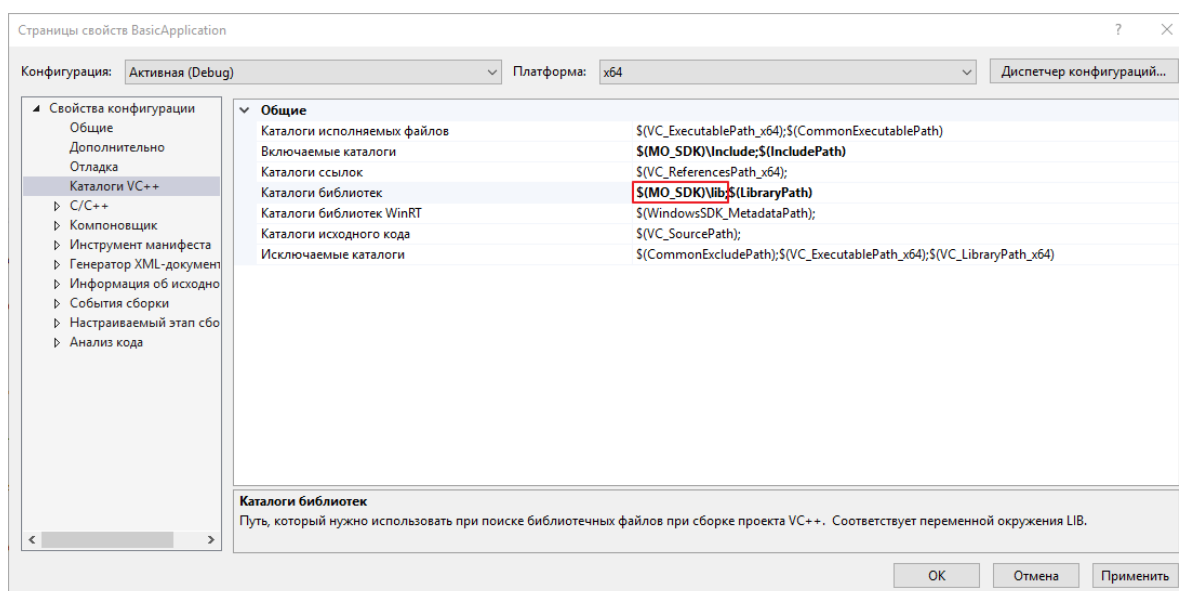


Рисунок 2 – Настройка каталогов библиотек

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
    
```

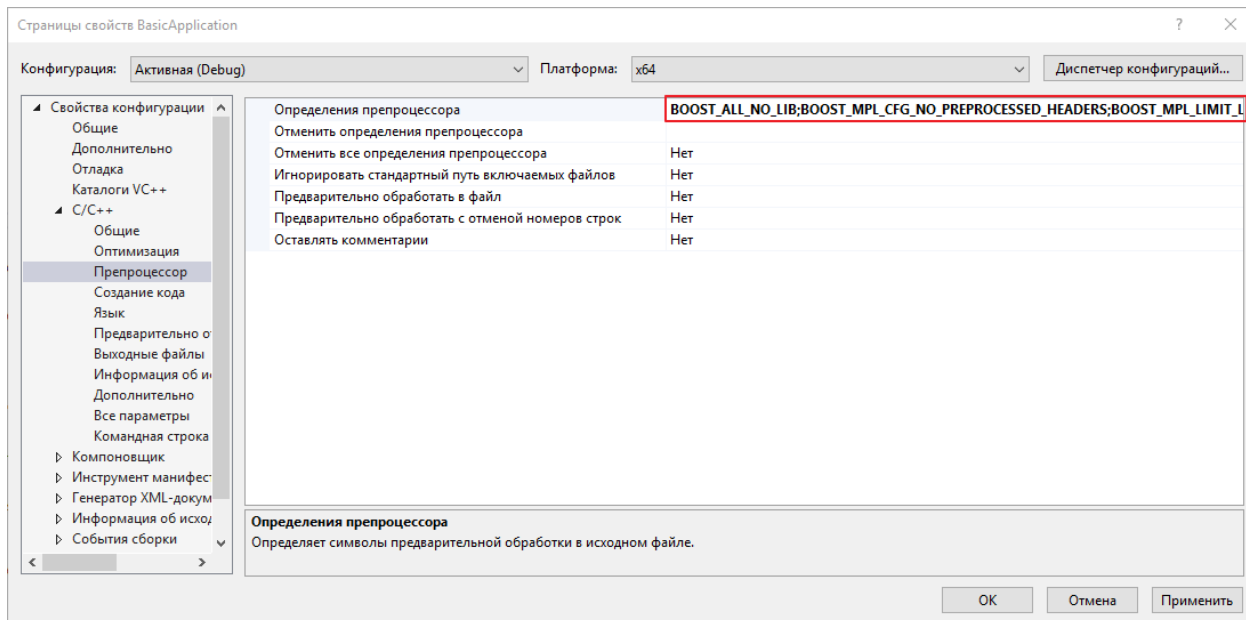


Рисунок 3 – Настройка определений препроцессора

4. Указать в конфигурации компоновщика библиотеку **MyOfficeDocumentAPI.lib** для конфигурации **Release** или **MyOfficeDocumentAPIId.lib** для конфигурации

Debug в качестве дополнительной зависимости (см. [Рисунок 4](#)). Данные библиотеки расположены в папке **MyOfficeDocumentAPIlib** каталога установки MyOffice Document API.

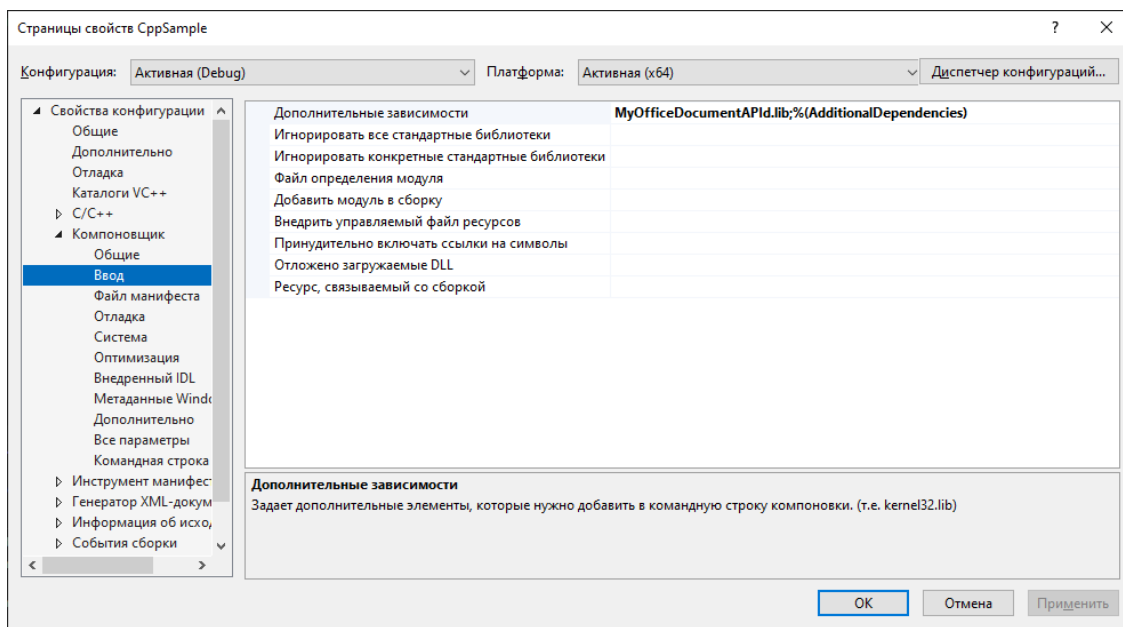


Рисунок 4 – Настройка дополнительных зависимостей

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 **MyOfficeDocumentAPI.dll** из папки **MyOfficeDocumentAPIlib** каталога установки MyOffice Document API в папку **c:\Project\BasicApplication\x64\Debug**.
2. Скопировать папку **Resources**, содержащую ресурсы приложения, из папки **MyOfficeDocumentAPIshare** 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:

- **сmake** – 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. Выполнить 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 **MyOfficeDocumentAPIId.dll** для конфигурации **Debug** или **MyOfficeDocumentAPI.dll** для конфигурации **Release** из папки **MyOfficeDocumentAPI\lib** каталога установки MyOffice Document API.

2. Папку **Resource**, содержащую ресурсы приложения, из папки **MyOfficeDocumentAPIshare** 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 **MyOfficeDocumentAPIshare\Resources** каталога установки MyOffice Document API).
2. Файл динамической библиотеки **MyOfficeDocumentAPIId.dll** для конфигурации **Debug** или **MyOfficeDocumentAPI.dll** для конфигурации **Release** (скопировать из папки **MyOfficeDocumentAPIlib** 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 ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

Приведенные примеры предназначены для ознакомления с основными функциями **MyOffice Document API**. Они демонстрируют использование классов, методов, структур и функций API при разработке приложений.

3.1 Проверка версии библиотеки

Перед использованием библиотеки рекомендуется проверить ее версию .

```
unsigned int ExpectedMajorAPIVersion = 1;
unsigned int ExpectedMinorAPIVersion = 0;

if (!Version::isAPIVersionCompatible(ExpectedMajorAPIVersion,
ExpectedMinorAPIVersion))
{
    // Сообщить о несовместимости версии и выйти.
}
// Начать работу с библиотекой.
```

3.2 Работа с текстовым документом

3.2.1 Создание и сохранение документа

Для создания документа необходимо получить экземпляр объекта типа `Application` и вызвать метод `createDocument`, который вернет новый документ указанного в параметре типа с настройками по умолчанию.

Возможен также вызов метода `createDocument` с использованием в качестве параметра заданных свойств создаваемого документа.

```
// Инициализация сессии редактирования
Application application;

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

Сохранение документа осуществляется с помощью метода `saveAs`, содержащегося в объектной модели документа.

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

// Сохранение документа в формате DOCX
auto outputFile = "BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been created." << std::endl;
```

3.2.2 Работа с текстом

3.2.2.1 Настройка свойств текста

Для настройки свойств текста необходимо создать объект `TextProperties` и инициализировать его поля.

```
// Вставка содержимого документа
document.getRange().getBegin().insertText("Hello! This is an example!");

// Доступ к тексту документа
auto range = document.getRange();

// Установка свойства текста
TextProperties textProperties;
textProperties.bold = true;
textProperties.textColor = ColorRGBA(255, 0, 0, 0);
range.setTextProperties(textProperties);

// Сохранение документа в формате DOCX
auto outputFile = "BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been created." << std::endl;
```

3.2.3 Работа с таблицами

3.2.3.1 Вставка таблицы

Для вставки таблицы необходимо задать ее положение в документе. В примере таблица вставляется в начало документа. В случае успешного выполнения вставки возвращается объект таблицы.

```
// Позиция начала документа
auto position = document.getRange().getBegin();

size_t rowsCount = 3;
size_t columnsCount = 3;
const auto tableName = "MyTable";

// Вставка таблицы
auto table = position.insertTable(rowsCount, columnsCount, tableName);

// Сохранение документа в формате DOCX
auto outputFile = "BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been updated." << std::endl;
```

3.2.3.2 Удаление таблицы

Для удаления таблицы предварительно нужно получить объект таблицы, который может быть найден по ее названию. Следует обратить внимание на то, что объект таблицы перестает существовать после удаления из документа.

```
// Вставка таблицы 3x3 в текстовый документ
auto position = document.getRange().getBegin();

size_t rowCount = 3;
size_t columnsCount = 3;
const auto tableName = "Table";
position.insertTable(rowCount, columnsCount, tableName);

// Удаление таблицы
auto table = document.getBlocks().getTable(tableName);
if (table) {
    table->remove();
}

// Сохранение документа в формате DOCX
auto outputFile = "BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been updated." << std::endl;
```

Объект таблицы также может быть получен из объекта document по индексу таблицы. Следует обратить внимание на то, что объект таблицы перестает существовать после удаления из документа.

```
// Удаление таблицы
auto table = document.getBlocks().getTable(0);
if (table) {
    table->remove();
}
```

3.2.4 Работа с ячейками таблицы

3.2.4.1 Объединение ячеек таблицы

Для объединения ячеек необходим объект [CellRange](#). Объект CellRange имеет методы объединения и разъединения ячеек.

```
auto cellRange = table.getCellRange(CellRangePosition(1,1,2,2));
cellRange.merge();
```

3.2.4.2 Разъединение ячеек таблицы

Для того чтобы разъединить ячейки, необходимо получить объект ячейки [Cell](#) из диапазона объединения ячеек. Объект Cell содержит метод для разъединения всех ячеек из диапазона, частью которого является эта ячейка.

```
auto cell = table.getCell(CellPosition(2,2));
cell.unmerge();
```

3.2.4.3 Поворот текста в ячейке

Для того чтобы задать поворот текста в ячейке, необходимо получить объект ячейки [CellProperties](#). Объект `CellProperties` содержит поле `textOrientation`, в котором содержится ориентация текста ячейки.

```
Document::CellProperties cellProps;
cellProps.textOrientation = Document::TextOrientation(90);
table->getCell("D10").setCellProperties(cellProps);
```

3.2.5 Форматирование таблицы

3.2.5.1 Установка свойств форматирования ячеек таблицы

Для форматирования ячеек таблицы необходимо задать набор ячеек для форматирования и применить к ним методы установки необходимых свойств.

```
CellProperties cellProperties;

cellProperties.verticalAlignment = VerticalAlignment::Center;

cellProperties.backgroundColor = ColorRGBA(211,211,211,1); // Light-Gray

auto tableEmployes = document.getBlocks().getTable(tableName);
if (tableEmployes) {
    auto cellOrder = tableEmployes->getCell(CellPosition(0,0));
    cellOrder.setText("№ n/n");
    cellOrder.setCellProperties(cellProperties);

    auto cellName = tableEmployes->getCell(CellPosition(0,1));
    cellName.setText("FIO");
    cellName.setCellProperties(cellProperties);
}
```

3.2.5.2 Установка границ ячеек таблицы

Для форматирования ячеек таблицы необходимо задать набор ячеек для форматирования и применить к ним методы установки необходимых свойств, например, настройки внешних и внутренних границ.

```
// Вставка таблицы 3x3
auto position = document.getRange().getBegin();

// Вставка таблицы
size_t rowCount = 3;
size_t columnsCount = 3;
const auto tableName = "MyTable";
auto tableEmployees = position.insertTable(rowCount, columnsCount, tableName);

// Установка внешних границ
auto cellRangeOuter = tableEmployees->getCellRange(CellRangePosition(0, 0, 2, 2));
```

```
LineProperties lineOuter;
lineOuter.style = LineStyle::Solid;
lineOuter.width = 2.0f;
lineOuter.color = ColorRGBA(255, 0, 255, 1);
Borders bordersOuter;
cellRangeOuter.setBorders(bordersOuter.setOuter(lineOuter));

// Установка внутренних границ
LineProperties lineInner;
lineInner.style = LineStyle::LongDash;
lineInner.width = 1.0f;
lineInner.color = ColorRGBA(128, 0, 128, 1);
Borders bordersInner;
cellRangeOuter.setBorders(bordersInner.setInnerHorizontal(lineInner));
cellRangeOuter.setBorders(bordersInner.setInnerVertical(lineInner));

// Сохранение документа в формате DOCX
auto outputFile = "./BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been updated." << std::endl;
```

3.2.6 Работа с закладками

3.2.6.1 Вставка закладки

Для вставки в документ закладки необходимо задать ее расположение. В примере приведено позиционирование закладки в начало документа. Наименование закладки должно быть уникальным.

```
// Вставка закладки в текстовый документ
auto bookmarkName = "Executor";
auto position = document.getRange().getBegin();
position.insertBookmark(bookmarkName);

// Сохранение документа в формате DOCX
auto outputFile = "./BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been updated." << std::endl;
```

3.2.6.2 Изменение содержимого закладки

Для изменения в документе закладки необходимо задать ее наименование. Наименование закладки должно быть уникальным.

```
// Вставка закладки в текстовый документ
auto bookmarkName = "Executor";
auto position = document.getRange().getBegin();
position.insertBookmark(bookmarkName);

// Изменение содержимого закладки
auto collectionBookmarks = document.getBookmarks();
auto rangeExecutor = collectionBookmarks.getBookmarkRange(bmkName);
auto txtExecutor = "Исполнитель: Иванов И.И., доб.1234";
```



```
rangeExecutor->replaceText(txtExecutor);

// Сохранение документа в формате DOCX
auto outputFile = "./BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been updated." << std::endl;
```

3.2.6.3 Удаление закладки

Для удаления закладки из документа используется метод объектной модели документа `removeBookmark(name)`, где `name` - наименование удаляемой закладки.

```
// Вставка закладки в текстовый документ
auto bookmarkName = "Executor";
auto position = document.getRange().getBegin();
position.insertBookmark(bookmarkName);

// Удаление закладки
document.removeBookmark(bmkName);

// Сохранение документа в формате DOCX
auto outputFile = "./BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been updated." << std::endl;
```

3.2.7 Работа с комментариями

3.2.7.1 Получение списка комментариев

```
// Получение коллекции комментариев
auto enumerator = document.getComments().getEnumerator();
while (enumerator->isValid())
{
    auto comment = enumerator->getCurrent();
    // Поля комментария
    // comment.getText()
    // derefOpt(comment.getInfo().author).name);
    // comment.getRange().extractText()
    enumerator->goToNext();
}
```

3.2.7.2 Получение списка ответов

```
// Получение коллекции комментариев
auto commentEnumerator = document.getRange().getComments().getEnumerator();
while (commentEnumerator->isValid())
{
    // Текущий комментарий
    auto comment = commentEnumerator->getCurrent();
    // Список ответов на комменатрий
    auto repliesEnumerator = comment.getReplies().getEnumerator();
    while (repliesEnumerator->isValid())
    {
        auto reply = enumerator->getCurrent();
    }
}
```

```
        // Поля ответа на комментария
        // reply.getText()
        // derefOpt(reply.getInfo().author).name
        repliesEnumerator->goToNext();
    }
    commentEnumerator->goToNext();
}
```

3.2.8 Экспорт текстового документа

Экспорт текстового документа в формат PDF/A-1.

```
// Наполнение документа.
document.getRange().getBegin().insertText("Hello! This is an example!");

// Экспорт в формат PDF
auto outputFile = "./BasicExample.pdf";
document.exportAs(outputFile, Document::ExportFormat::PDFa1);

std::cout << "Done: the '" << outputFile << "' file has been created." << std::endl;
```

3.2.9 Поиск в текстовом документе

Для поиска в текстовом документе необходимо с помощью глобальной функции `createSearch` создать экземпляр объекта `Search` и вызвать метод `findText`.

```
// Вставка содержимого документа
auto text = "API documentation describes what services an API offers and how \
to use those services, aiming to cover everything a client would \
need to know for practical purposes. Documentation is crucial for \
the development and maintenance of applications using the API. API \
\
documentation is traditionally found in documentation files but \
can also be found in social media such as blogs, forums, and Q&A \
websites.";
document.getRange().getBegin().insertText(text);

// Поиск текстового фрагмента в документе
auto textSearch = createSearch(document);
auto collect = textSearch->findText("API");

// Установка свойства текста для найденных фрагментов
TextProperties textProperties;
textProperties.bold = true;

while (collect->isValid()) {
    auto rng = collect->getCurrent();

    // Установка начертания "полужирный"
    rng.setTextProperties(textProperties);
    collect->goToNext();
}

// Сохранение документа в формате DOCX
```

```
auto outputFile = "./BasicExample.docx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been created." << std::endl;
```

3.2.10 Управление ориентацией и свойствами страниц раздела

Для установки ориентации страницы можно использовать метод `setPageOrientation` объекта `section`, который может быть получен из блока документа.

```
auto section = document.getBlocks().getBlock(0)->getSection();
section.setPageOrientation(PageOrientation::Portrait);
```

Установить необходимые значения высоты и ширины страниц раздела документа можно с помощью метода `setPageProperties`, задав необходимые значения в структуре `PageProperties`.

```
PageProperties properties;
properties.width = 350.f;
properties.height = 800.f;

auto section = document.getBlocks().getBlock(0)->getSection();
section.setPageProperties(properties);
```

Ориентация страниц может быть установлена для каждого раздела документа. Список разделов документа может быть получен из объекта `document`.

```
auto sectionsEnumerator = document.getSections().getEnumerator();
while (sectionsEnumerator->isValid())
{
    auto section = sectionsEnumerator->getCurrent();
    section.setPageOrientation(PageOrientation::Portrait);
    sectionsEnumerator->goToNext();
}
```

Ориентация страниц объекта `section` может быть получена с использованием метода `getPageOrientation`.

```
const auto section = document.getBlocks().getParagraph(0)->getSection();
const auto orientation = section.getPageOrientation();
```

Свойства страниц объекта `section` могут быть получены с использованием метода `getPageProperties`.

```
const auto section = document.getBlocks().getParagraph(0)->getSection();
const auto properties = section.getPageProperties();
```

3.2.11 Работа с отслеживаемыми изменениями

```
// Получение коллекции отслеживаемых изменений из всего диапазона документа
auto enumerator = document.getRange().getTrackedChangesEnumerator();
while (enumerator->isValid())
{
```

```
// Получение одного изменения
auto trackedChange = enumerator->getCurrent();
// Поля структуры TrackedChange:
// trackedChange.getType();
// derefOpt(trackedChange.getInfo().author).name);
// trackedChange.getRange().extractText();
enumerator->goToNext();
}
```

3.2.12 Работа с колонтитулами раздела

Для получения колонтитулов раздела следует использовать методы `getHeaders` или `getFooters` объекта `section`.

```
// Исходный документ содержит несколько разделов (sections). На каждой
// странице присутствует верхний (header) и нижний (footer) колонтитул.

// Первый раздел (section) в документе
auto section = document.getSections().getEnumerator()->getCurrent();

// Верхний колонтитул первого раздела
auto header = section.getHeaders().getEnumerator()->getCurrent();

// Нижний колонтитул первого раздела
auto footer = section.getFooters().getEnumerator()->getCurrent();

std::cout << header.getRange().extractText();
std::cout << footer.getRange().extractText();
```

3.3 Работа с табличным документом

3.3.1 Создание и сохранение документа

Для создания документа необходимо получить экземпляр объекта типа [Application](#) и вызвать метод `createDocument`, который вернет новый документ указанного в параметре типа с настройками по умолчанию. Возможен также вызов метода `createDocument` с использованием в качестве параметра заданных свойств создаваемого документа.

```
// Инициализация сессии
Application application;

Document::DocumentSettings createSettings;
createSettings.documentType = Document::DocumentType::Workbook;
createSettings.formulaType = Document::FormulaType::R1C1;

// Создание табличного документа
auto document = application.createDocument(createSettings);
```

Сохранение документа осуществляется с помощью метода `saveAs`, содержащегося в объектной модели документа.

```
auto position = document.getRange().getEnd();
auto table = position.insertTable(30, 30, "Sheet");

// Заполнение документа
table.getCell(Document::CellPosition(1,1)).setNumber(1);
table.getCell(Document::CellPosition(2,2)).setNumber(2);
table.getCell(Document::CellPosition(3,3)).setNumber(3);

// Сохранение документа
Document::SaveDocumentSettings saveSettings;
saveSettings.documentType = Document::DocumentType::Workbook;
saveSettings.documentFormat = Document::DocumentFormat::OXML;
auto outputFile = "BasicExample.xlsx";
document.saveAs(outputFile, saveSettings);
```

3.3.2 Работа с текстом

3.3.2.1 Настройка свойств текста

Для настройки свойств текста необходимо создать объект [TextProperties](#) и инициализировать его поля.

```
TextProperties textProperties;
textProperties.bold = true;
textProperties.textColor = ColorRGBA(255, 0, 0, 0);
auto para = document.getBlocks().getParagraph(0);
auto range = para->getRange();
range.setTextProperties(textProperties);
```

3.3.3 Работа с листами табличного документа

3.3.3.1 Вставка рабочего листа в табличный документ

Для вставки листа табличного документа необходимо задать позицию вставки. В примере в качестве позиции используется конец документа. Объект таблицы будет возвращен в случае успешного завершения операции вставки.

```
auto position = document.getRange().getEnd();
auto table = position.insertTable(10, 10, "Sheet");
```

3.3.3.2 Удаление рабочего листа табличного документа

Для удаления листа документа его нужно предварительно получить по названию. Следует обратить внимание на то, что объект листа перестает существовать после удаления из табличного документа.

```
auto table = document.getBlocks().getTable(sheetName);
if (table) {
    table->remove();
}
```

Объект листа также может быть получен из объекта `document` по индексу. Индексация листов начинается с нуля. Необходимо обратить внимание на то, что объект листа перестает существовать после удаления из документа.

```
auto table = document.getBlocks().getTable(0);
if (table) {
    table->remove();
}
```

3.3.4 Работа с ячейками таблицы

3.3.4.1 Настройка свойств ячейки

Для настройки свойств ячейки необходимо создать объект [CellProperties](#) и инициализировать необходимые поля.

```
CellProperties cellProperties;
cellProperties.verticalAlignment = VerticalAlignment::Center;
cellProperties.backgroundColor = ColorRGBA(255, 0, 0, 0);
cell.setCellProperties(cellProperties);
```

Пример задания угла поворота текста в ячейке:

```
// Наполнение документа.
auto position = document.getRange().getEnd();

size_t rowCount = 26;
size_t columnsCount = 10;
const auto tableName = "MyTable";
auto table = position.insertTable(rowCount, columnsCount, tableName);

// Вставка содержимого документа
auto txtD10 = "API documentation describes what services an API offers\n"
             "and how to use those services, aiming to cover everything \n"
             "a client would need to know for practical purposes.";

table.getCell("D10").setText(txtD10);

// Поворот текста в ячейке на 90 градусов
Document::CellProperties cellProps;
cellProps.textOrientation = Document::TextOrientation(90);
table.getCell("D10").setCellProperties(cellProps);

// Сохранение документа в формате XLSX
auto outputFile = "BasicExample.xlsx";
document.saveAs(outputFile);
```

Пример установки выравнивания текста абзаца по ширине ячейки:

```
size_t rowCount = 26;
size_t columnsCount = 10;
const auto tableName = "MyTable";
auto table = position.insertTable(rowCount, columnsCount, tableName);

// Вставка содержимого документа
auto txtD10 = "API documentation describes what services an API offers\n and how
to use those services, "
```

```
        "aiming to cover everything \na client would need to know for
practical purposes.";

table.getCell("D10").setText(txtD10);

// Выравнивание текста по ширине ячейки
Document::ParagraphProperties paraProps;
paraProps.alignment = Document::Alignment::Justify;
table.getCell("D10").setParagraphProperties(paraProps);

// Сохранение документа в формате XLSX
auto outputFile = "BasicExample.xlsx";
document.saveAs(outputFile);
```

3.3.4.2 Объединение ячеек таблицы

Для объединения ячеек необходимо получить объект [CellRange](#), который обладает методами объединения и разъединения ячеек.

```
auto cellRange = table.getCellRange(CellRangePosition(1, 1, 2, 2));
derefPtr(cellRange).merge();
```

3.3.4.3 Разъединение ячеек таблицы

Для того чтобы разъединить ячейки, необходимо получить объект ячейки [Cell](#) из диапазона объединения ячеек. Объект `Cell` содержит метод для разъединения всех ячеек из диапазона, частью которого является эта ячейка.

```
auto cell = table.getCell(CellPosition(2, 2));
cell.unmerge();
```

3.3.5 Экспорт табличного документа

Экспорт табличного документа в формат PDF/A-1.

```
auto position = document.getRange().getEnd();
size_t rowsCount = 26;
size_t columnsCount = 10;
const auto tableName = "MyTable";

// Наполнение документа.
auto table = position.insertTable(rowsCount, columnsCount, tableName);

// Вставка содержимого документа
auto txtA1 = "API documentation describes what services an API "
            "offers and how to use those services, aiming to "
            "cover everything a client would need to know for "
            "practical purposes.";
auto txtA2 = "Documentation is crucial for the development and "
            "maintenance of applications using the API.";
auto txtA3 = "API documentation is traditionally found in "
            "documentation files but can also be found in social "
            "media such as blogs, forums, and Q&A websites.";

table.getCell("A1").setText(txtA1);
```

```
table.getCell("A2").setText(txtA2);
table.getCell("A3").setText(txtA3);

// Экспорт в формат PDF
auto outputFile = "../BasicExample.pdf";
document.exportAs(outputFile, Document::ExportFormat::PDFa1);

std::cout << "Done: the '" << outputFile << "' file has been created." << std::endl;
```

3.3.6 Поиск в табличном документе

Для поиска в табличном документе необходимо с помощью глобальной функции `createSearch` создать для документа экземпляр объекта [Search](#) и вызвать метод `findText`.

```
// Получение позиции конца документа
auto position = document.getRange().getEnd();

size_t rowCount = 26;
size_t columnsCount = 10;
const auto tableName = "MyTable";
auto table = position.insertTable(rowCount, columnsCount, tableName);

// Вставка содержимого документа
auto txtA1 = "API documentation describes what services an API offers "
            "and how to use those services, aiming to cover "
            "everything a client would need to know for practical "
            "purposes.";
auto txtA2 = "Documentation is crucial for the development and "
            "maintenance of applications using the API.";
auto txtA3 = "API documentation is traditionally found in "
            "documentation files but can also be found in "
            "social media such as blogs, forums, and Q&A websites.";

table.getCell("A1").setText(txtA1);
table.getCell("A2").setText(txtA2);
table.getCell("A3").setText(txtA3);

// Поиск текстового фрагмента в документе
auto textSearch = createSearch(document);
auto collect = textSearch->findText("API");

// Установка свойства текста для найденных фрагментов
TextProperties textProperties;
textProperties.bold = true;

while (collect->isValid()) {
    auto rng = collect->getCurrent();

    // Установка начертания "полужирный"
    rng.setTextProperties(textProperties);
    collect->goToNext();
}

// Сохранение документа в формате XLSX
```



```
auto outputFile = "../BasicExample.xlsx";
document.saveAs(outputFile);

std::cout << "Done: the '" << outputFile << "' file has been created." << std::endl;
```

3.3.7 Работа со сводными таблицами

3.3.7.1 Получение диапазона исходных данных сводной таблицы

```
// Исходный документ содержит сводную таблицу
// Получаем ячейку, находящуюся в диапазоне исходных данных сводной таблицы
const auto pivotRootCell = document.getBlocks().getTable(1)-
>getCell(CellPosition(2, 0));

// Получаем сводную таблицу
auto pivotTable = pivotRootCell.getPivotTable();

// Получаем диапазон исходных данных сводной таблицы
auto sourceCellRange = pivotTable->getSourceRange();

// Для получения границ диапазона используем поля CellRange:
// sourceCellRange.getBeginRow()
// sourceCellRange.getBeginColumn()
// sourceCellRange.getLastRow()
// sourceCellRange.getLastColumn()
```

3.3.7.2 Получение диапазона размещения сводной таблицы

```
// Получаем диапазон размещения сводной таблицы
auto pivotCellRange = pivotTable->getPivotRange();
```

3.3.7.3 Получение неподдерживаемых свойств сводной таблицы

```
// Получаем неподдерживаемые свойства сводной таблицы
auto pivotUnsupportedFeatures = pivotTable->getUnsupportdeFeatures();
```

3.3.7.4 Получение флагов отображения общих итогов для строк и колонок

```
// Получаем флаги отображения общих итогов для строк и колонок
bool isRowGrandTotalEnabled = pivotTable->isRowGrandTotalEnabled();
bool isColGrandTotalEnabled = pivotTable->isColumnGrandTotalEnabled();
```

3.3.7.5 Получение заголовков сводной таблицы

```
// Получение заголовков сводной таблицы
PivotTableCaptions captions = pivotTable->getPivotTableCaptions();

// Используем поля структуры PivotTableCaptions:
// captions.emptyCaption.get()
// captions.errorCaption.get()
// captions.rowHeaderCaption
// captions.columnHeaderCaption
// captions.valuesHeaderCaption
```

3.3.7.6 Получение и применение фильтра для сводной таблицы

```
// По названию поля сводной таблицы получаем фильтр
auto filter = pivotTable->getFilter("Category");

// 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();
```

3.3.7.7 Получение полей из области фильтров

```
// Получение полей из области фильтров
PivotTablePageFields pageFields = pivotTable->getPageFields();
// Перебираем все поля из области фильтров
for (size_t i = 0; i < pageFields.size(); ++i)
{
    const auto& fieldProps = pageFields[i].fieldProperties;

    // Далее используем поля структуры PivotTableFieldProperties:
    // fieldProps.fieldName
    // fieldProps.fieldAlias
    // fieldProps.subtotalAlias
}
```

3.3.7.8 Получение полей из области значений

```
// Получение полей из области значений
PivotTableValueFields valueFields = pivotTable->getValueFields();
// Перебираем все поля из области значений
for (size_t i = 0; i < valueFields.size(); ++i)
{
    const auto& valueField = valueFields[i];

    // Далее используем поля структуры PivotTableValueField:
    // valueField.baseFieldName
    // valueField.valueFieldName
    // valueField.cellNumberFormat
    // valueField.totalFunction
    // valueField.customFormula
}
```

3.3.7.9 Получение полей из области строк

```
// Получение полей из области строк
PivotTableCategoryFields rowFields = pivotTable->getRowFields();
// Перебираем все поля из области строк
for (size_t i = 0; i < rowFields.size(); ++i)
{
    const auto& fieldProperties = rowFields[i].fieldProperties;
    const auto& subtotalFunctions = rowFields[i].subtotalFunctions;

    // Далее используем поля структуры PivotTableCategoryField:
    // fieldProperties.fieldName
    // fieldProperties.fieldAlias
    // fieldProperties.subtotalAlias
}
```

3.3.7.10 Получение полей из области колонок

```
// Получение полей из области колонок
PivotTableCategoryFields columnFields = pivotTable->getColumnFields();
// Перебираем все поля из области колонок
for (size_t i = 0; i < columnFields.size(); ++i)
```

```
{
    const auto& fieldProperties = columnFields[i].fieldProperties;
    const auto& subtotalFunctions = columnFields [i].subtotalFunctions;

    // Далее используем поля структуры PivotTableCategoryField:
    // fieldProperties.fieldName
    // fieldProperties.fieldAlias
    // fieldProperties.subtotalAlias
}
```

3.3.7.11 Получение настроек отображения сводной таблицы

```
PivotTableLayoutSettings layoutSettings =
    pivotTable->getPivotTableLayoutSettings();

// Далее используем поля структуры PivotTableLayoutSettings:
// layoutSettings.reportLayout
// layoutSettings.pageFieldOrder
// layoutSettings.useGridDropZones
// layoutSettings.pageFieldWrapCount
// layoutSettings.displayFieldCaptions
// layoutSettings.indentForCompactLayout
// layoutSettings.valueFieldsOrientation
// layoutSettings.isMergeAndCenterLabelsEnabled
```

3.3.7.12 Обновление сводной таблицы

```
// Пересчет и перезаполнение сводной таблицы в соответствии с исходными
данными.
// Обновление сводной таблицы приводит к потере всех неподдерживаемых свойств.
pivotTable->update();
```

3.3.8 Работа с именованными выражениями

3.3.8.1 Получение именованного выражения

```
// Исходный документ содержит именованное выражение
// Получаем именованное выражение с именем "Age_of_Majority"
auto ageOfMajority = document.getNamedExpressions().get("Age_of_Majority");
```

3.3.8.2 Получение именованного выражения таблицы

```
// Получаем именованное выражение таблицы с именем "Alice_Age"
auto aliceAge = document.getBlocks().getTable(0)->
    getNamedExpressions().get("Alice_Age");
```

3.3.8.3 Получение свойств именованного выражения

```
// Получаем именованное выражение с именем "Alice_Age"
auto expression = document.getBlocks().getTable(0)->
    getNamedExpressions().get("Alice_Age");
// Далее используем поля структуры NamedExpression:
auto name = expression->getName();
auto formula = expression->getExpression();
auto range = expression->getCellRange();
```

3.3.8.4 Получение коллекции именованных выражений

```
// Получение коллекции именованных выражений
auto enumerator = document.getBlocks().getTable(0)->
    getNamedExpressions().getEnumerator();
```

```
// Перебор коллекции именованных выражений
for (; enumerator->isValid(); enumerator->goToNext())
{
    auto current = enumerator->getCurrent();
    // Использование полей структуры NamedExpression
    // current.getName()
    // current.getExpression()
    // current.getCellRange()
}
```

3.4 Работа со встроенными объектами

Доступ ко встроенным объектам документа осуществляется посредством использования метода `getInlineObjects`.

```
auto enumerator = document.getRange().getInlineObjects().GetEnumerator();
while (enumerator->isValid())
{
    auto inlineObject = enumerator->getCurrent();

    // Проверка типа встроенного объекта
    auto image = inlineObject.toImage();
    if (image)
    {
        // Использование встроенного объекта типа Image
    }

    enumerator->goToNext();
}
```

С помощью объекта типа `Frame` 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.
auto frame = inlineObject.getFrame();
if (auto wrapType = frame.getWrapType())
{
    if (*wrapType == TextWrapType::Inline)
    {
        frame.setWrapType(TextWrapType::TopAndBottom);
    }
}
```

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

```
auto frame = inlineObject.getFrame();

TextAnchoredPosition position;

HorizontalTextAnchoredPosition
    horizontalPosition{HorizontalRelativeTo::Page, 12.f};
```

```
VerticalTextAnchoredPosition
    verticalPosition{VerticalRelativeTo::PageTopMargin, 122.f};

position.horizontal = horizontalPosition;
position.vertical = verticalPosition;

frame.setPosition(position);
```

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

```
auto frame = inlineObject.getFrame();

TextAnchoredPosition position;

HorizontalTextAnchoredPosition
    horizontalPosition{HorizontalRelativeTo::Page,
        HorizontalAnchorAlignment::Center};

VerticalTextAnchoredPosition
    verticalPosition{VerticalRelativeTo::PageTopMargin,
        VerticalAnchorAlignment::Top};

position.horizontal = horizontalPosition;
position.vertical = verticalPosition;

frame.setPosition(position);
```

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

```
auto frame = inlineObject.getFrame();

TextAnchoredPosition position;

position.horizontal =
    HorizontalTextAnchoredPosition{HorizontalRelativeTo::Column, 125.f};
position.vertical =
    VerticalTextAnchoredPosition{VerticalRelativeTo::Page, 345.f};

frame.setPosition(position);
```

С помощью метода `Frame::setDimensions` можно изменить размеры встроенных объектов

```
auto frame = inlineObject.getFrame();

Size<Unit> size{300.f, 400.f};
frame.setDimensions(size);
```

4 СПРАВОЧНИК КЛАССОВ, СТРУКТУР И МЕТОДОВ

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

4.1 Типы документов

4.1.1 Класс DocumentType

Тип DocumentType описывает поддерживаемые типы документов.

```
enum class DocumentType
{
    Text,
    Workbook,
    Presentation
};
```

Поддерживаемые типы документов:

- Text – используется для работы с текстовыми документами – файлы DOCX, ODT, XODT, TXT;
- Workbook – используется для работы с табличными документами – файлы XLSX, ODS, XODS;
- Presentation – используется для работы с презентационными документами – файлы PPTX, ODP. Работа с презентационными документами средствами Document API в настоящий момент не поддерживается.

4.2 Форматы документов

4.2.1 Класс DocumentFormat

Тип `DocumentFormat` определяет поддерживаемые форматы документов.

```
enum class DocumentFormat
{
    PlainText,
    DSV,
    OXML,
    ODF,
    HTML,
    PDF,
    PDFA
};
```

Поддерживаемые форматы документов:

- `PlainText` – используется для работы с файлами TXT;
- `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;
- `OXML` – используется для работы с текстовыми (DOCX) или табличными (XLSX) документами в формате Open Office XML;
- `ODF` – используется для работы с текстовыми (ODT) или табличными (ODS) документами формата Open Document Format (ГОСТ Р ИСО/МЭК 26300-2010);
- `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;
- `PDF` – используется для работы с документами в формате Portable Document Format (PDF), версии 1.4. Средствами Document API поддерживается только операция экспорта документа в формат PDF/A-1b;
- `PDFA` – используется для работы с документами в формате Portable Document Format (PDF) для долгосрочного архивного хранения (PDF/A-1b). Средствами Document API поддерживается только операция экспорта документа в формат PDF/A-1b.

При работе с файлами XODT или XODS (MyOffice eXtended Open Document Format) используйте значение `DocumentFormat::ODF`.

Средствами MyOffice Document API не поддерживается работа с текстовыми и табличными документами в двоичном формате Microsoft Word (DOC) и Microsoft Excel (XLS, XLSB), а также документами, содержащими макрокоманды VBA (DOCM, XLSM).

4.3 Форматы экспорта документов

4.3.1 Класс ExportFormat

Тип ExportFormat определяет поддерживаемые форматы документов для экспорта.

```
enum class ExportFormat
{
    PDFA1
};
```

Формат PDFA1 используется для работы с документами в формате Portable Document Format (PDF) для долгосрочного архивного хранения (PDF/A-1b).

Средствами Document API поддерживается только операция экспорта документа в формат PDF/A-1b.

4.4 Неподдерживаемые свойства документа

Имеются свойства документа, которые могут быть утеряны при сохранении документа.

4.4.1 Структура SaveUnsupportedFeatures

Структура SaveUnsupportedFeatures представлена как STL контейнер `std::set`, содержащий флаги типа `SaveUnsupportedFeature`.

```
using SaveUnsupportedFeatures = std::set<SaveUnsupportedFeature>;
enum class SaveUnsupportedFeature
{
    CommentsInHeaderFooter,
    CommentsInFootnote,
    MixedNoteAlignment,
};
```

Флаги `SaveUnsupportedFeature` описывают следующие свойства документа:

- `CommentsInHeaderFooter` – комментарии в колонтитулах;
- `CommentsInFootnote` – комментарии в сносках;
- `MixedNoteAlignment` – параграфы с разным выравниванием в заметках.

4.5 Системы адресации ячеек

4.5.1 Класс FormulaType

Тип FormulaType определяет поддерживаемые системы адресации ячеек (стили ссылок) в табличном документе.

```
enum class FormulaType
{
    A1,
    R1C1
};
```

Вариант A1 соответствует наиболее распространенной системе адресации ячеек, при которой столбцы задаются буквами, а строки – числами.

Вариант R1C1 соответствует альтернативной системе адресации ячеек, при которой столбцы и строки задаются числами.

4.6 Кодировки документов

4.6.1 Класс Encoding

Тип Encoding определяет поддерживаемые кодировки документов.

```
enum class Encoding
{
    Unknown,
    UTF8,
    UTF16BE,
    UTF16LE,
    UTF32BE,
    UTF32LE,
    Windows1250,
    Windows1251,
    Windows1252,
    ISO8859Part5,
    KOI8R,
    KOI8U,
    CP866
};
```

4.7 Типы выравнивания текста

4.7.1 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.

4.7.2 Класс TextLayout

Тип TextLayout содержит варианты форматирования текста в ячейке таблицы.

```
enum class TextLayout
{
    SingleLine,
    WrapByWords,
    ShrinkSizeToFitWidth
};
```

Варианты форматирования текста:

- SingleLine – текст располагается в одной строке (если текст длиннее объекта, содержащего его, то текст просто перекрывается с соседними объектами);

- WrapByWords – текст выравнивается по высоте объекта, содержащего его, при этом высота объекта может быть увеличена до необходимого размера;
- ShrinkSizeToFitWidth – размер текста выбирается таким образом, что текст соответствует ширине объекта, в котором он находится.

4.7.3 Класс VerticalAlignment

Тип VerticalAlignment содержит варианты выравнивания текста в ячейке по вертикали.

```
enum class VerticalAlignment
{
    Bottom,
    Center,
    Top
};
```

Варианты выравнивания текста в ячейке по вертикали представлены в [Таблице 3](#).

Таблица 3 – Варианты выравнивания текста в ячейке по вертикали

Тип выравнивания	Представление в интерфейсе	
Bottom		
Center		

Тип выравнивания	Представление в интерфейсе	
Тор		

4.7.4 Класс TextWrapType

Тип TextWrapType содержит варианты обтекания текстом встроенного объекта (изображения, формы и т. д.).

```
enum class TextWrapType
{
    Inline,
    InFrontOfText,
    BehindText,
    TopAndBottom,
    Square,
    Through,
    Tight
};
```

Варианты обтекания текстом:

- Inline – встроенный объект располагается в тексте;
- InFrontOfText – встроенный объект располагается перед текстом;
- BehindText – встроенный объект располагается за текстом;
- TopAndBottom – текст располагается сверху и снизу встроенного объекта;
- Square – текст располагается вокруг прямоугольной рамки встроенного объекта;
- Through – текст обтекает встроенный объект по сторонам и внутри;
- Tight – текст обтекает встроенный объект по сторонам.

4.8 Стили линий

4.8.1 Класс LineStyle

Тип `LineStyle` содержит доступные стили линий для оформления границ ячейки.

```
enum class LineStyle
{
    NoLine,
    Solid,
    Dot,
    Dash,
    LongDash,
    DashDot,
    DotDotDash,
    Double,
    Wave
};
```

Варианты стилей линий:

- `NoLine` – нет границ;
- `Solid` – обычная линия;
- `Dot` – пунктирная линия;
- `Dash` – штриховая линия;
- `LongDash` – разомкнутая линия;
- `DashDot` – штрихпунктирная линия;
- `DotDotDash` – штрихпунктирная линия с двумя точками;
- `Double` – двойная линия;
- `Wave` – волнистая линия.

4.9 Типы надстрочного и подстрочного форматирования

4.9.1 Класс ScriptPosition

Тип `ScriptPosition` содержит варианты форматирования отдельного символа в тексте – надстрочный знак, подстрочный знак или обычный символ.

```
enum class ScriptPosition
{
    SuperScript,
    SubScript,
    NormalScript
};
```

4.10 Типы форматов ячеек

4.10.1 Класс CellFormat

Класс CellFormat содержит поддерживаемые форматы ячеек таблицы.

```
enum class CellFormat
{
    General,
    Percentage,
    Number,
    Text,
    Currency,
    Accounting,
    Date,
    Time,
    DateTime,
    Fraction,
    Scientific,
    Custom
};
```

Варианты форматов ячеек:

- General – общий, например: 12;
- Percentage – процентный, например: 120,00%;
- Number – числовой, например: 12,00;
- Text – текстовый, например: 12;
- Currency – денежный, например: 12 000,00Р;
- Accounting – финансовый, например: 12 000,00Р;
- Date – дата, например: 01.01.2020;
- Time – время, например: 0:00:00;
- DateTime – дата и время, например: 12.12.2020 0:00:00;
- Fraction – дробный, например: 12 1/5;
- Scientific – экспоненциальный, например: 1,22E+01;
- Custom – пользовательский.

4.10.2 Структура AccountingCellFormatting

Структура AccountingCellFormatting содержит параметры для финансового формата ячеек таблицы.

```
struct AccountingCellFormatting
{
    std::uint8_t decimalPlaces = 2;
    boost::optional<std::string> symbol;
```

```
boost::optional<std::uint32_t> localeCode;  
boost::optional<std::string> fillSymbol;  
bool useThousandsSeparator = true;  
CurrencySignPlacement currencySignPlacement{};  
};
```

Параметры финансового формата ячейки:

- `decimalPlaces` – количество десятичных позиций;
- `symbol` – символ денежной единицы;
- `localeCode` – идентификатор кода языка (MS-LCID);
- `fillSymbol` – символ заполнения;
- `useThousandsSeparator` – использовать разделитель для тысячных;
- `currencySignPlacement` – тип размещения знака валюты [CurrencySignPlacement](#).

4.10.3 Структура PercentageCellFormatting

Структура `PercentageCellFormatting` содержит параметр для процентного формата ячеек таблицы.

```
struct PercentageCellFormatting  
{  
    uint8_t decimalPlaces = 2;  
};
```

Параметр настройки процентного формата ячейки:

- `decimalPlaces` – количество десятичных позиций.

4.10.4 Структура NumberCellFormatting

Структура `NumberCellFormatting` содержит параметры для числового формата ячеек таблицы.

```
struct NumberCellFormatting  
{  
    uint8_t decimalPlaces = 2;  
    bool useThousandsSeparator = false;  
    bool useRedForNegative = false;  
    bool useBracketsForNegative = false;  
    bool hideSign = false;  
};
```

Параметры числового формата ячейки:

- `decimalPlaces` – количество десятичных позиций;
- `useThousandsSeparator` – использовать разделитель для тысячных;
- `useRedForNegative` – использовать красный цвет для отрицательных значений;

- `useBracketsForNegative` – использовать скобки для отрицательных значений;
- `hideSign` – скрывать знак «минус» для отрицательных значений.

4.10.5 Структура `CurrencyCellFormatting`

Структура `CurrencyCellFormatting` содержит параметры для денежного формата ячеек таблицы.

```
struct CurrencyCellFormatting
{
    uint8_t decimalPlaces = 2;
    boost::optional<std::string> symbol;
    boost::optional<std::uint32_t> localeCode;
    bool useThousandsSeparator = false;
    bool useRedForNegative = false;
    bool useBracketsForNegative = false;
    bool hideSign = false;
};
```

Параметры денежного формата ячейки:

- `decimalPlaces` – количество десятичных позиций;
- `symbol` – символ денежной единицы;
- `localeCode` – идентификатор кода языка (MS-LCID);
- `useThousandsSeparator` – использовать разделитель для тысячных;
- `useRedForNegative` – использовать красный цвет для отрицательных значений;
- `useBracketsForNegative` – использовать скобки для отрицательных значений;
- `hideSign` – скрывать знак «минус» для отрицательных значений.

4.10.6 Структура `DateTimeCellFormatting`

Структура `DateTimeCellFormatting` содержит параметры для формата ячеек таблицы типа Дата и Время.

```
struct DateTimeCellFormatting
{
    DatePatterns dateListID{};
    TimePatterns timeListID{};
};
```

Параметры формата ячейки:

- `dateListID` – формат даты;
- `timeListId` – формат времени.

4.10.6.1 Класс DatePatterns

Класс DatePatterns содержит перечисление форматов даты.

```
enum class DatePatterns
{
    DayMonthTextLongYearLong,
    FullDate,
    DayMonthNumberLongYearLong,
    DayMonthNumberLongYearShort,
    DayMonthNumberShortYearShort,
    DayMonthTextShort,
    DayMonthTextShortYearShort,
    DayMonthYearLongNonLocalizableHyphen,
    DayMonthYearLongNonLocalizableSlash,
};
```

Форматы даты:

- DayMonthTextLongYearLong – 'mmmm dd, yyyy' для языка en_US;
- FullDate – 'день недели, mmmm dd, yyyy' для языка en_US;
- DayMonthNumberLongYearLong – 'mm/dd/yyyy' для языка en_US;
- DayMonthNumberLongYearShort – 'm/dd/yy' для языка en_US;
- DayMonthNumberShortYearShort – 'dd-mmm' для языка en_US;
- DayMonthTextShort – 'mmm-yy' для языка en_US;
- DayMonthTextShortYearShort – 'mmm dd, yy' для языка en_US;
- DayMonthYearLongNonLocalizableHyphen – нелокализуемый шаблон 'dd-mm-yyyy';
- DayMonthYearLongNonLocalizableSlash – нелокализуемый шаблон 'dd/mm/yyyy'.

4.10.6.2 Класс TimePatterns

Класс TimePatterns содержит перечисление форматов времени.

```
enum class TimePatterns
{
    ShortTime,
    LongTime,
};
```

Форматы времени:

- ShortTime – 'hh:mm AM/PM' для языка en_US;
- LongTime – 'hh:mm:ss AM/PM' для языка en_US.

4.10.7 Структура FractionCellFormatting

Структура FractionCellFormatting содержит параметры для дробного формата ячеек таблицы.

```
struct FractionCellFormatting
{
    boost::optional<std::uint32_t> minNumeratorDigits;
    boost::optional<std::uint32_t> minDenominatorDigits;
    boost::optional<std::uint32_t> denominatorValue;
};
```

Параметры дробного формата ячейки:

- minNumeratorDigits – количество позиций числителя;
- minDenominatorDigits – количество позиций знаменателя;
- denominatorValue – знаменатель.

4.10.8 Структура ScientificCellFormatting

Структура ScientificCellFormatting содержит параметры для экспоненциального формата ячеек таблицы.

```
struct ScientificCellFormatting
{
    std::uint8_t decimalPlaces = 2;
    std::uint8_t minExponentDigits = 2;
};
```

Параметры экспоненциального формата ячейки:

- decimalPlaces – количество десятичных позиций;
- minExponentDigits – минимальное количество позиций экспоненты.

4.11 Типы межстрочного интервала

4.11.1 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.

4.12 Типы схем форматирования списков


4.12.1 Класс ListSchema







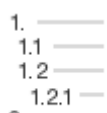
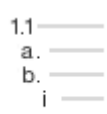
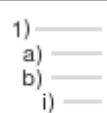
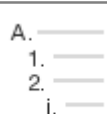
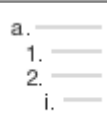
Класс ListSchema содержит типы схем форматирования списков, которые могут быть применены к абзацам текста.

```
enum class ListSchema
{
    Unknown,
    UnknownBullet,
    UnknownNumbering,
    BulletCircleSolid,
    BulletCircleContour,
    BulletSquareSolid,
    BulletDiamondDots,
    BulletHyphen,
    BulletConcaveArrowSolid,
    BulletCheckmark,
    EnumeratorDecimalDot,
    EnumeratorDecimalDotMultiLevel,
    EnumeratorDecimalBracket,
    EnumeratorLatinUppercaseDot,
    EnumeratorLatinLowercaseDot,
    EnumeratorLatinLowercaseBracket,
    EnumeratorRomanUppercaseDot,
    EnumeratorRomanLowercaseDot,
    EnumeratorDecimalRussianBracket,
    EnumeratorRussianLowercaseBracket
};
```

Типы схем форматирования списков представлены в [Таблице 4](#).

Таблица 4 – Типы 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	

Тип схемы списка	Описание типа схемы списка	Изображение
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	
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	

Тип схемы списка	Описание типа схемы списка	Изображение
EnumeratorLatinLowercaseBracket	Нумерация латинскими строчными буквами со скобкой	a) _____ 1) _____ 2) _____ i) _____ b) _____
EnumeratorRomanUppercaseDot	Нумерация римскими прописными цифрами с точкой	I. _____ a. _____ b. _____ 1. _____ II. _____
EnumeratorRomanLowercaseDot	Нумерация римскими строчными цифрами с точкой	i. _____ 1. _____ 2. _____ i. _____ ii. _____
EnumeratorDecimalRussianBracket	Десятичная нумерация через запятую со скобкой	1) _____ a) _____ б) _____ i) _____ 2) _____
EnumeratorRussianLowercaseBracket	Нумерация с русскими строчными буквами со скобкой	a) _____ 1) _____ 2) _____ i) _____ б) _____

4.13 Типы отслеживаемых изменений

4.13.1 Класс TrackedChangeType

Класс TrackedChangeType содержит типы отслеживаемых изменений.

```
enum class TrackedChangeType
{
    Added,
    Removed
};
```

Типы отслеживаемых изменений:

- Added – добавленные изменения;
- Removed – удаленные изменения.

4.14 Типы колонтитулов

4.14.1 Класс HeaderFooterType

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

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

4.15 Масштабирование при печати табличных документов

4.15.1 Класс WorksheetPrinterFitType

Класс `WorksheetPrinterFitType` содержит варианты масштабирования при печати табличных документов.

```
enum class WorksheetPrinterFitType : std::uint8_t
{
    ActualSize,
    ByPageScale,
    ByPageBreaksOnly,
    FitToPage,
    FitToWidth,
    FitToHeight
};
```

Варианты масштабирования:

- `ActualSize` – фактический размер;
- `ByPageScale` – по масштабу страницы;
- `ByPageBreaksOnly` – по разрыву страниц;
- `FitToPage` – вписать в страницу;
- `FitToWidth` – вписать по ширине;
- `FitToHeight` – вписать по высоте.

4.16 Выбор страниц для экспорта и печати

4.16.1 Класс PageParity

Класс `PageParity` содержит варианты отбора страниц для экспорта, печати документов.

```
enum class PageParity
{
    Odd,
    Even,
    All
};
```

Типы страниц:

- `Odd` – печать только нечетных страниц;
- `Even` – печать только четных страниц;
- `All` – печать всех страниц.

4.17 Типы переменных

4.17.1 Тип Percents

Тип Percents используется для измерения безразмерных величин, например, масштаб и т. д.

```
using Percents = float;
```

4.17.2 Тип SheetNames

Тип SheetNames используется в качестве контейнера имен листов для печати, экспорта и т. д.

```
using SheetNames = std::vector<std::string>;
```

4.17.3 Тип HorizontalTextAnchoredPosition

Тип HorizontalTextAnchoredPosition используется для управления расположением объекта относительно закрепленной позиции по горизонтали.

```
using HorizontalTextAnchoredPosition =  
RelativeAnchoredPosition<HorizontalTypeTraits>;
```

4.17.4 Тип VerticalTextAnchoredPosition

Тип VerticalTextAnchoredPosition используется для управления расположением объекта относительно закрепленной позиции по вертикали.

```
using VerticalTextAnchoredPosition =  
RelativeAnchoredPosition<VerticalTypeTraits>;
```

4.17.5 Тип HorizontalTypeTraits

Тип HorizontalTypeTraits используется для управления расположением объекта относительно закрепленной позиции по горизонтали.

```
struct HorizontalTypeTraits  
{  
    using RelativeToType = HorizontalRelativeTo;  
    using AlignmentType = HorizontalAnchorAlignment;  
};
```

4.17.6 Тип VerticalTypeTraits

Тип VerticalTypeTraits используется для управления расположением объекта относительно закрепленной позиции по вертикали.

```
struct VerticalTypeTraits  
{  
    using RelativeToType = VerticalRelativeTo;  
    using AlignmentType = VerticalAnchorAlignment;  
};
```

4.18 Типы идентификаторов цветов тем

4.18.1 Класс ThemeColorID

Класс ThemeColorID содержит идентификаторы цветов темы.

```
enum class ThemeColorID
{
    Background1,
    Text1,
    Background2,
    Text2,
    Dark1,
    Dark2,
    Light1,
    Light2,
    Accent1,
    Accent2,
    Accent3,
    Accent4,
    Accent5,
    Accent6,
    Hyperlink,
    FollowedHyperlink
};
```

Типы идентификаторов цветов тем:

- Background1 – Фон1, значение по умолчанию 0;
- Text1 – Текст1;
- Background2 – Фон2;
- Text2 – Текст2;
- Dark1 – Темная1;
- Dark2 – Темная2;
- Light1 – Светлая1;
- Light2 – Светлая2;
- Accent1 – Акцент1;
- Accent2 – Акцент2;
- Accent3 – Акцент3;
- Accent4 – Акцент4;
- Accent5 – Акцент5;
- Accent6 – Акцент6;
- Hyperlink – Гиперссылка;
- FollowedHyperlink – Следующая гиперссылка.

4.19 Типы окончаний линий






4.19.1 Класс LineEndingStyle

Класс LineEndingStyle содержит типы окончаний линий.

```
enum class LineEndingStyle
{
    Arrow,
    Diamond,
    Oval,
    Stealth,
    Triangle,
    None
};
```

Типы окончания линий представлены в [Таблице 5](#).

Таблица 5 – Типы окончания линий

Тип окончания линии	Изображение
Arrow	
Diamond	
Oval	
Stealth	
Triangle	
None	

4.20 Типы размещения объекта по горизонтали

4.20.1 Класс HorizontalRelativeTo

Класс HorizontalRelativeTo содержит все типы размещения объекта относительно закрепленной позиции по горизонтали.

```
enum class HorizontalRelativeTo
{
    Character,
    Column,
    ColumnLeftMargin,
    ColumnRightMargin,
    ColumnInsideMargin,
    ColumnOutsideMargin,
    Page,
    PageContent,
    PageLeftMargin,
    PageRightMargin,
};
```

```
PageInsideMargin,  
PageOutsideMargin  
};
```

Типы характеристик:

- Character – символ;
- Column – столбец;
- ColumnLeftMargin – левое поле столбца;
- ColumnRightMargin – правое поле столбца;
- ColumnInsideMargin – внутреннее поле столбца;
- ColumnOutsideMargin – внешнее поле столбца;
- Page – страница;
- PageContent – содержимое страницы;
- PageLeftMargin – левое поле страницы;
- PageRightMargin – правое поле страницы;
- PageInsideMargin – внутреннее поле страницы;
- PageOutsideMargin – внешнее поле страницы.

4.21 Типы размещения объекта по вертикали

4.21.1 Класс VerticalRelativeTo

Класс VerticalRelativeTo содержит все типы размещения объекта относительно закрепленной позиции по вертикали.

```
enum class VerticalRelativeTo  
{  
    Character,  
    BaseLine,  
    Paragraph,  
    Page,  
    PageContent,  
    PageTopMargin,  
    PageBottomMargin,  
    PageInsideMargin,  
    PageOutsideMargin  
};
```

Типы характеристик:

- Character – символ;
- BaseLine – базовая линия;
- Paragraph – абзац;
- Page – страница;

- PageContent – содержимое страницы;
- PageTopLeftMargin – верхнее поле страницы;
- PageBottomMargin – нижнее поле страницы;
- PageInsideMargin – внутреннее поле страницы;
- PageOutsideMargin – внешнее поле страницы.

4.22 Типы выравнивания объекта по вертикали

4.22.1 Класс VerticalAnchorAlignment

Класс VerticalAnchorAlignment содержит все типы выравнивания объекта относительно закрепленной позиции по вертикали.

```
enum class VerticalAnchorAlignment
{
    Top,
    Bottom,
    Center,
    Inside,
    Outside
};
```

Типы выравнивания:

- Top – по верхнему краю;
- Bottom – по нижнему краю;
- Center – по центру;
- Inside, Outside – по границам.

4.23 Типы выравнивания объекта по горизонтали

4.23.1 Класс HorizontalAnchorAlignment

Класс HorizontalAnchorAlignment содержит все типы выравнивания объекта относительно закрепленной позиции по горизонтали.

```
enum class HorizontalAnchorAlignment
{
    Left,
    Right,
    Center,
    Inside,
    Outside
};
```

Типы выравнивания:

- Left – по левому краю;
- Right – по правому краю;

- Center – по центру;
- Inside, Outside – по границам.

4.24 Классы и структуры, относящиеся к ядру (Core Types)

4.24.1 Класс Application

Класс `Application` управляет параметрами и объектами приложения. Предоставляет интерфейс для создания и загрузки документов. Как правило, создается только один объект класса для всего сеанса работы с документом

```
class Application
{
public:
    Application(boost::optional<std::string> resourcePath = boost::none);

    std::shared_ptr<const Messenger> getMessenger() const;
    std::shared_ptr<Messenger> getMessenger();

    Document::Document createDocument(Document::DocumentType documentType);
    Document::Document createDocument(const Document::DocumentSettings& settings);

    Document::Document loadDocument(const std::string& filePath);
    Document::Document loadDocument(const std::string& filePath,
                                    const Document::LoadDocumentSettings& settings);
};
```

4.24.1.1 Метод Application

Конструктор класса `Application`. Получает путь к ресурсам приложения. Если путь не указан, поиск ресурсов производится в подкаталоге `Resources` текущего каталога. При невозможности создания экземпляра класса вызывает исключение [ApplicationCreateError](#).

```
Application(boost::optional<std::string> resourcePath = boost::none);
```

4.24.1.2 Метод getMessenger

Методы `getMessenger` возвращают объект `Messenger`, реализующий логирование событий.

```
std::shared_ptr<const Messenger> getMessenger() const;
std::shared_ptr<Messenger> getMessenger();
```

4.24.1.3 Метод createDocument

Создает новый документ указанного в параметрах типа с настройками по умолчанию или с указанными в параметре `settings`. Список поддерживаемых типов документов приведен в разделе [DocumentType](#). Настройки документа приведены в разделе [DocumentSettings](#).

При невозможности создания документа вызывается исключение

[DocumentCreateError](#).

```
Document::Document createDocument(Document::DocumentType documentType);  
Document::Document createDocument(const Document::DocumentSettings& settings);
```

4.24.1.4 Метод loadDocument

Загружает существующий текстовый или табличный документ из файла, находящегося по указанному пути `filePath`. Формат и тип документа определяются из расширения файла, если они не указаны явно с помощью параметра `settings`. Описание структуры [LoadDocumentSettings](#).

При невозможности загрузки файла вызывает исключение [DocumentLoadError](#).

```
Document::Document loadDocument(const std::string& filePath);  
Document::Document loadDocument(const std::string& filePath,  
                                const Document::LoadDocumentSettings& settings);
```

4.24.2 Класс Messenger

Класс `Messenger` предоставляет возможность работы по логированию событий.

```
class Messenger  
{  
public:  
    typedef std::function<void(const Message&)> MessageHandlerFunction;  
    std::shared_ptr<Connection> subscribe(const std::shared_ptr<MessageHandler>  
handler);  
    std::shared_ptr<Connection> subscribe(const MessageHandlerFunction&  
handler);  
    void notify(const Message& message);  
};
```

4.24.2.1 Метод subscribe

Методы `subscribe` служат для подписки на события лога.

```
std::shared_ptr<Connection> subscribe(const std::shared_ptr<MessageHandler>  
handler);  
std::shared_ptr<Connection> subscribe(const MessageHandlerFunction& handler);
```

4.24.2.2 Метод notify

Метод `notify` используется для создания события лога.

```
void notify(const Message& message);
```

4.24.3 Класс Connection

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

```
class Connection  
{
```

```
public:  
    void unsubscribe();  
};
```

4.24.4 Класс Message

Класс Message предназначен для формирования событий лога. Внутренний класс Severity описывает уровни сообщений лога (информация, предупреждение, ошибка).

```
class Message  
{  
public:  
    enum class Severity : std::uint8_t  
    {  
        Info,  
        Warning,  
        Error  
    };  
    bool operator==(const Message& other) const;  
  
    Severity getSeverity() const;  
    const std::string& getText() const;  
    static Message makeInfo(const std::string& text);  
    static Message makeWarning(const std::string& text);  
    static Message makeError(const std::string& text);  
};
```

4.24.4.1 Оператор ==

Оператор сравнения == используется для определения эквивалентности значений двух сообщений лога.

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

4.24.4.2 Оператор !=

Оператор сравнения != используется для определения неэквивалентности значений двух сообщений лога.

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

4.24.4.3 Метод getSeverity

Метод возвращает уровень лога (Info, Warning, Error).

```
Severity getSeverity() const;
```

4.24.4.4 Метод getText

Метод возвращает текст сообщения.

```
const std::string& getText() const;
```

4.24.4.5 Метод `makeInfo`

Метод создает сообщение типа `Info` с заданным текстом.

```
static Message makeInfo(const std::string& text);
```

4.24.4.6 Метод `makeWarning`

Метод создает сообщение типа `Warning` с заданным текстом.

```
static Message makeWarning(const std::string& text);
```

4.24.4.7 Метод `makeError`

Метод создает сообщение типа `Error` с заданным текстом.

```
static Message makeError(const std::string& text);
```

4.24.5 Класс `MessageHandler`

Класс предназначен для обработки сообщений лога, метод `handle` вызывается при возникновении сообщения.

```
class MessageHandler
{
public:
    virtual void handle(const Message& message) const = 0;
};
```

4.24.6 Структура `DocumentSettings`

Структура `DocumentSettings` предоставляет настройки, необходимые на протяжении всего периода работы с документом.

```
struct DocumentSettings
{
    UserInfo userInfo;
    LocaleInfo localeInfo;
    TimeZone timeZone;
    FormulaType formulaType = FormulaType::A1;
    DocumentType documentType;
};
```

Структура `DocumentSettings` включает следующие поля:

- `userInfo` – информация о пользователе [UserInfo](#);
- `documentType` – тип документа [DocumentType](#);
- `localeInfo` – информация о локализации [LocaleInfo](#);
- `timeZone` – информация о временной зоне [TimeZone](#);
- `formulaType` – система адресации ячеек [FormulaType](#).

4.24.7 Структура LoadDocumentSettings

Структура LoadDocumentSettings предоставляет дополнительные настройки, необходимые для загрузки документа из файла.

```
struct LoadDocumentSettings
{
    DocumentSettings commonDocumentSettings;
    Encoding encoding = Encoding::UTF8;
    DSVSettings dsvSettings;
    boost::optional<std::string> documentPassword;
};
```

Структура **LoadDocumentSettings** включает следующие поля:

- commonDocumentSettings – общие настройки документа [DocumentSettings](#);
- encoding – кодировка документа [Encoding](#);
- dsvSettings – настройки, необходимые для работы с файлами CSV (comma-separated value) и DSV (delimiter-separated value) (см. раздел [DSVSettings](#));
- documentPassword – пароль для защиты электронного документа от несанкционированного доступа. Механизм парольной защиты поддерживается только для семейства ОС Microsoft Windows.

4.24.8 Структура SaveDocumentSettings

Структура SaveDocumentSettings предоставляет дополнительные настройки, необходимые для сохранения документа в файл.

```
struct SaveDocumentSettings
{
    boost::optional<DocumentFormat> documentFormat;
    boost::optional<DocumentType> documentType;
    boost::optional<std::string> documentPassword;
    bool isTemplate = false;
    DSVSettings dsvSettings = {};
};
```

Структура **SaveDocumentSettings** включает следующие поля:

- documentFormat – формат документа [DocumentFormat](#);
- documentType – тип документа [DocumentType](#);
- documentPassword – пароль для защиты электронного документа от несанкционированного доступа;
- isTemplate – флаг, обозначающий, что документ должен быть сохранен как шаблон;
- dsvSettings – структура, необходимая для сохранения в формате DSV.

4.24.9 Структура UserInfo

Структура `UserInfo` предоставляет информацию о пользователе.

```
struct UserInfo
{
    std::string name;
    std::string email;
};
```

Структура `UserInfo` включает следующие поля:

- `name` – имя пользователя;
- `email` – почтовый адрес пользователя.

4.24.10 Класс CurrencySignPlacement

Тип `CurrencySignPlacement` определяет варианты размещения знака валюты, до значения (`$12.00`), либо после (`12,00 P`).

```
enum class CurrencySignPlacement
{
    Prefix,
    Suffix
};
```

Класс `CurrencySignPlacement` включает следующие свойства:

- `Prefix` – знак валюты, размещаемый до значения;
- `Suffix` – знак валюты, размещаемый после значения.

4.24.11 Структура LocaleInfo

Структура `LocaleInfo` предоставляет информацию о локализации.

```
struct LocaleInfo
{
    std::string localeName = "en_US";
    boost::optional<std::string> decimalSeparator;
    boost::optional<std::string> thousandSeparator;
    boost::optional<std::string> listSeparator;
    boost::optional<std::string> currencySymbol;
    boost::optional<CurrencySignPlacement> currencyFormat;
    boost::optional<std::string> shortDatePattern;
    boost::optional<std::string> longDatePattern;
    boost::optional<std::string> timePattern;
};
```

Структура `LocaleInfo` включает следующие поля:

- `localeName` – название локализации представлено в формате `<language><REGION>`, где языковой код соответствует стандарту ISO-639, а код региона стандарту ISO-3166, значение по умолчанию - «`en_US`»;

- `decimalSeparator` – десятичный разделитель, отделяющий целые и дробные части числа;
- `thousandSeparator` – символ, разделяющий группы цифр в числовых значениях;
- `listSeparator` – разделитель, который используется для разделения элементов списка;
- `currencySymbol` – символ валюты;
- `currencyFormat` – расположение знака валюты в текущем регионе;
- `shortDatePattern` – заданный «короткий» формат отображения даты в текущем регионе (например, 'm/d/yy' для en_US);
- `longDatePattern` – заданный «длинный» формат отображения даты в текущем регионе (например, 'dddd, mmmm d, yyyy' для en_US);
- `timePattern` – заданный формат отображения времени в текущем регионе (например, 'h:mm AM/PM' для en_US).

4.24.12 Структура `TimeZone`

Структура `TimeZone` предоставляет информацию о часовом поясе.

```
struct TimeZone
{
    int32_t offsetInSecondsToUTC = 0;
};
```

Структура `TimeZone` содержит поле `offsetInSecondsToUTC` с помощью которого задается смещение или разность между временем в данном часовом поясе и временем в формате UTC (Всемирное координированное время).

4.24.13 Структура `DSVSettings`

Структура `DSVSettings` предоставляет настройки, необходимые для работы с файлами CSV (comma-separated value) и DSV (delimiter-separated value).

```
struct DSVSettings
{
    std::u32string separators = U",";
    char32_t escapeChar = '\'';
    bool autofit = false;
    size_t startBlockIndex = 0;
    size_t lastBlockIndex = 0;
};
```

Структура `DSVSettings` включает следующие поля:

- `separators` – символ-разделитель полей данных, значение по умолчанию – запятая;
- `escapeChar` – символ-разделитель для текстовых строк, значение по умолчанию – двойная кавычка;
- `autofit` – признак необходимости автоматического подстраивания ширины столбца под размер данных в ячейке, значение по умолчанию – `false`;
- `startBlockIndex` – поле содержит индекс блока документа для начала сохранения;
- `endBlockIndex` – поле содержит индекс блока документа для окончания сохранения.

4.24.14 Структура `Point`

Структура `Point` представляет собой точку в двухмерном пространстве.

```
struct Point
{
    Point();
    Point(const T& xArg, const T& yArg);

    std::string toString() const;

    T x;
    T y;
};
```

4.24.14.1 Метод `Point`

Конструктор по умолчанию.

```
Point();
```

Конструктор класса устанавливает положение точки в двухмерном пространстве.

```
Point(const T& xArg, const T& yArg);
```

4.24.14.2 Метод `toString`

Возвращает информацию о координатах точки в виде строкового значения формата `(x: <value>, y: <value>)`.

```
std::string toString() const;
```

4.24.15 Структура `Size`

Структура `Size` описывает размер объекта в двухмерном пространстве.

```
struct Size
{
    Size();
```

```
Size(const T& widthArg, const T& heightArg);

std::string toString() const;

T width;
T height;
};
```

4.24.15.1 Метод Size

Конструктор по умолчанию.

```
Size();
```

Конструктор класса устанавливает размер объекта в двухмерном пространстве.

```
Size(const T& widthArg, const T& heightArg);
```

4.24.15.2 Метод toString

Возвращает информацию о размерах объекта в виде строкового значения формата (width: <value>, height: <value>).

```
std::string toString() const;
```

4.24.16 Структура Rect

Структура Rect описывает прямоугольник.

```
struct
{
    Rect();
    Rect(const Point<T>& topLeftArg,
        const Point<T>& bottomRightArg);
    Rect(const T& topLeftX, const T& topLeftY,
        const T& bottomRightX, const T& bottomRightY);
    std::string toString() const;
    Point<T> topLeft;
    Point<T> bottomRight;
};
```

4.24.16.1 Метод Rect

Конструктор по умолчанию.

```
Rect();
```

Конструктор класса устанавливает положение и размер прямоугольника в двухмерном пространстве.

```
Rect(const Point<T>& topLeftArg,
    const Point<T>& bottomRightArg);
Rect(const T& topLeftX, const T& topLeftY,
    const T& bottomRightX, const T& bottomRightY);
```

4.24.16.2 Метод toString

Возвращает информацию о положении и размерах прямоугольника в виде строкового значения формата (topLeft: <value>, bottomRight: <value>).

```
std::string toString() const;
```

4.24.17 Структура ColorRGBA

Структура ColorRGBA позволяет настроить пользовательский цвет для оформления элементов документа, текста, границ таблиц и т. п. Для описания цвета используется расширенная цветовая модель RGB, позволяющая установить непрозрачность цвета.

```
struct ColorRGBA
{
    ColorRGBA();
    ColorRGBA(const std::uint8_t rArg,
              const std::uint8_t gArg,
              const std::uint8_t bArg,
              const std::uint8_t aArg);
    bool operator==(const ColorRGBA& other) const;
    bool operator!=(const ColorRGBA& other) const;
    std::uint8_t r;
    std::uint8_t g;
    std::uint8_t b;
    std::uint8_t a;
};
```

4.24.17.1 Метод ColorRGBA

Конструктор по умолчанию. По умолчанию компоненты красного, синего и зеленого цветов устанавливаются в значение 0 (черный цвет), значение прозрачности равно 0 (абсолютно прозрачный).

```
ColorRGBA();
```

Конструктор с параметрами устанавливает заданные значения для компонентов красного, синего и зеленого цветов, а также прозрачности.

```
ColorRGBA(const std::uint8_t rArg, const std::uint8_t gArg,
          const std::uint8_t bArg, const std::uint8_t aArg);
```

4.24.17.2 Оператор ==

Оператор сравнения == используется для определения эквивалентности значений двух цветовых моделей, включая значение прозрачности.

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

4.24.17.3 Оператор !=

Оператор сравнения != используется для определения неэквивалентности значений

двух цветовых моделей, включая значение прозрачности.

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

4.24.17.4 Поля структуры ColorRGBA

Поля r, g, b, a позволяют получить прямой доступ для чтения или записи значений компонентов красного, синего и зеленого цветов, а также значения прозрачности.

4.24.18 Структура DateTime

Структура DateTime предоставляет дату и время с точностью до секунды.

```
struct DateTime
{
    uint16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
    bool operator==(const DateTime& other) const
    bool operator!=(const DateTime& other) const
};
```

4.24.18.1 Поля структуры DateTime

Структура DateTime содержит следующие поля:

- year – год;
- month – месяц;
- day – день;
- hour – часы;
- minute – минуты;
- second – секунды.

4.24.18.2 Оператор ==

Оператор сравнения == используется для определения эквивалентности двух значений времени.

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

4.24.18.3 Оператор !=

Оператор сравнения != используется для определения неэквивалентности двух значений времени.

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

4.25 Классы, структуры и методы объектной модели документа

4.25.1 Класс Document

Класс Document предоставляет доступ к объектной модели документа и используется для управления содержимым текстового или табличного документа.

```
class Document
{
public:
    void saveAs(const std::string& filePath) const;
    void saveAs(const std::string& filePath,
                const SaveDocumentSettings& settings) const;

    void exportAs(const std::string& filePath, ExportFormat format) const;

    void exportAs(const std::string& filePath, ExportFormat format,
                  const TextExportSettings& settings) const;
    void exportAs(const std::string& filePath, ExportFormat format,
                  const WorkbookExportSettings& settings) const;

    SaveUnsupportedFeatures validate(DocumentFormat format,
                                     DocumentType type) const;

    void setChangesTrackingEnabled(bool value);
    bool isChangesTrackingEnabled() const;

    void setMirroredMarginsEnabled(bool value);
    bool areMirroredMarginsEnabled() const;

    void setPageProperties(const PageProperties& properties);
    void setPageOrientation(PageOrientation orientation);

    Blocks getBlocks();

    Bookmarks getBookmarks();

    Comments getComments();

    NamedExpressions getNamedExpressions();

    std::shared_ptr<Scripts> getScripts();

    Range getRange();

    Sections getSections();
    Document merge(Document& other);

    boost::optional<PivotTablesManager> getPivotTablesManager();
};
```

4.25.1.1 Метод saveAs

Сохраняет документ в файл по указанному пути. Формат и тип документа определяются расширением файла, если не указаны в явном виде.

При невозможности сохранения документа возникает исключение

[DocumentSaveError](#).

```
void saveAs(const std::string& filePath) const;
void saveAs(const std::string& filePath,
            const SaveDocumentSettings& settings) const;
```

4.25.1.2 Метод `exportAs`

Экспортирует документ в файл по указанному пути и с указанным форматом. В настоящее время поддерживается только операция экспорта документа в формат PDF/A-1b.

При невозможности экспорта документа возникает исключение

[DocumentExportError](#).

```
void exportAs(const std::string& filePath, ExportFormat format) const;
```

Расширенные версии метода позволяют задать дополнительные настройки экспорта документа (например, страницы для экспорта и т. д.).

Для текстовых документов используется структура [TextExportSettings](#):

```
void exportAs(const std::string& filePath, ExportFormat format,
            const TextExportSettings& settings) const;
```

Для табличных документов используется структура [WorkbookExportSettings](#):

```
void exportAs(const std::string& filePath, ExportFormat format,
            const WorkbookExportSettings& settings) const;
```

ⓘ **Внимание !** Скрытые листы в табличном документе не экспортируются.

4.25.1.3 Метод `validate`

Данный метод проверяет документ и возвращает структуру [SaveUnsupportedFeatures](#), содержащую сведения о свойствах, которые могут быть потеряны при сохранении документа заданного типа [DocumentType](#) в заданный формат [DocumentFormat](#).

```
SaveUnsupportedFeatures validate(DocumentFormat format,
                                DocumentType type) const;
```

4.25.1.4 Метод `getBlocks`

Обеспечивает доступ к блокам данных, содержащихся в документе.

```
Blocks getBlocks();
```

4.25.1.5 Метод `getBookmarks`

Обеспечивает доступ к коллекции закладок (`bookmark`) в документе.

```
Bookmarks getBookmarks();
```


4.25.1.6 Метод `getNamedExpressions`

Используется для получения списка именованных выражений [NamedExpressions](#).

```
NamedExpressions getNamedExpressions();
```

4.25.1.7 Метод `getScripts`

Обеспечивает доступ к коллекции макрокоманд в документе.

```
std::shared_ptr<Scripts> getScripts();
```

4.25.1.8 Метод `getRange`

Возвращает диапазон, содержащий весь документ.

```
Range getRange();
```

4.25.1.9 Метод `getComments`

Обеспечивает доступ к комментариям [Comments](#), которые хранятся в документе.

```
Comments getComments();
```

4.25.1.10 Метод `setChangesTrackingEnabled`

Устанавливает включение/выключение отслеживания изменений в документе.

```
void setChangesTrackingEnabled(bool value);
```

4.25.1.11 Метод `isChangesTrackingEnabled`

Возвращает состояние включения отслеживания изменений в документе.

```
bool isChangesTrackingEnabled() const;
```

4.25.1.12 Метод `merge`

Возвращает документ, в котором различия отмечены отслеживаемыми изменениями.

```
Document merge(Document& other);
```

4.25.1.13 Метод `setPageProperties`

Устанавливает ширину и высоту страниц в документе (см. [PageProperties](#)).

```
void setPageProperties(const PageProperties& properties);
```

4.25.1.14 Метод `setPageOrientation`

Устанавливает альбомную, либо книжную ориентацию страниц в документе (см. [PageOrientation](#)).

```
void setPageOrientation(PageOrientation orientation);
```

4.25.1.15 Метод `getSections`

Используется для доступа к разделам документа (см. [Sections](#)).

```
Sections getSections();
```

4.25.1.16 Метод `setMirroredMarginsEnabled`

Метод позволяет включать/отключать зеркальные поля в документе.

```
void setMirroredMarginsEnabled(bool value);
```

4.25.1.17 Метод `areMirroredMarginsEnabled`

Возвращает состояние включения зеркальных полей в документе.

```
bool areMirroredMarginsEnabled() const;
```

4.25.1.18 Метод `getPivotTablesManager`

Возвращает [PivotTablesManager](#), который используется для создания сводных таблиц.

```
boost::optional<PivotTablesManager> getPivotTablesManager();
```

4.25.2 Класс `Table`

Класс `Table` предоставляет доступ к листу электронной таблицы или отдельной таблице в составе текстового документа.

```
class Table : public Block
{
public:

    std::string getName() const;
    void setName(const std::string& newName);

    std::size_t getRowCount() const;
    std::size_t getColumnCount() const;

    void insertColumnAfter(size_t columnIndex,
                          bool copyColumnStyle = true,
                          size_t columnsCount = 1);
    void insertColumnBefore(size_t columnIndex,
                            bool copyColumnStyle = true,
                            size_t columnsCount = 1);
    void insertRowAfter(size_t rowIndex,
                       bool copyRowStyle = true,
                       size_t rowsCount = 1);
    void insertRowBefore(size_t rowIndex,
                        bool copyRowStyle = true,
                        size_t rowsCount = 1);

    void removeColumn(size_t columnIndex,
                     size_t columnsCount = 1);
    void removeRow(size_t rowIndex,
```

```

        size_t rowCount = 1);

void setColumnsVisible(size_t first, size_t columnsCount, bool visible);
void setRowsVisible(size_t first, size_t rowCount, bool visible);

void groupRows(size_t first, size_t rowCount);
void ungroupRows(size_t first, size_t rowCount);
void clearRowGroups(size_t first, size_t rowCount);
void groupColumns(size_t first, size_t columnsCount);
void ungroupColumns(size_t first, size_t columnsCount);
void clearColumnGroups(size_t first, size_t columnsCount);

void setColumnWidth(size_t columnIndex, float width);
void setRowHeight(size_t rowIndex, float height,
                  boost::optional<RowHeightRule> heightRule);

CellRange getCellRange(const std::string& range);
CellRange getCellRange(const CellRangePosition& range);

Cell getCell(const std::string& cellRef);
Cell getCell(const CellPosition& position);

Charts getCharts();

NamedExpressions getNamedExpressions();

void setVisible(bool visible);
bool isVisible() const;

void duplicate();

void moveTo(size_t index);

void setPrintArea(const boost::optional<CellRangePosition>& range);
void setPrintAreas(const CellRangePositions& ranges);
CellRangePositions getPrintAreas() const;

bool operator==(const Table& other) const;
bool operator!=(const Table& other) const;
};

```

4.25.2.1 Наименование таблицы

Метод `getName` позволяет получить наименование листа табличного документа.

```
std::string getName() const;
```

Метод `setName` позволяет установить значение `newName` в качестве наименования листа табличного документа. Данное значение должно быть уникальным, т.к. может использоваться для ссылки на таблицу, например, из формул.

```
void setName(const std::string& newName);
```

4.25.2.2 Размеры таблицы

Методы `getRowCount` и `getColumnCount` позволяют получить количество строк и столбцов на листе табличного документа или для отдельной таблицы в составе текстового документа.

```
std::size_t getRowCount() const;
std::size_t getColumnCount() const;
```

4.25.2.3 Управление столбцами и строками

Методы `insertColumnBefore` и `insertColumnAfter` позволяют добавить в таблицу новые столбцы в количестве `columnsCount`, до или после столбца с индексом `columnIndex`. Индексация столбцов начинается с нуля.

Параметр `copyColumnStyle` указывает, следует ли использовать для новых столбцов элементы оформления столбца `columnIndex`.

```
void insertColumnAfter(size_t columnIndex,
                      bool copyColumnStyle = true,
                      size_t columnsCount = 1);
void insertColumnBefore(size_t columnIndex,
                       bool copyColumnStyle = true,
                       size_t columnsCount = 1);
```

Методы `insertRowBefore` и `insertRowAfter` позволяют добавить в таблицу новые строки в количестве `rowCount` до или после строки с индексом `rowIndex`. Индексация строк начинается с нуля.

Параметр `copyRowStyle` указывает, следует ли использовать для новых строк элементы оформления строки `rowIndex`.

```
void insertRowAfter(size_t rowIndex,
                   bool copyRowStyle = true,
                   size_t rowCount = 1);
void insertRowBefore(size_t rowIndex,
                    bool copyRowStyle = true,
                    size_t rowCount = 1);
```

Метод `removeColumn` позволяет удалить столбцы в количестве `columnsCount` начиная с индекса `columnIndex`. Индексация столбцов начинается с нуля.

```
void removeColumn(size_t columnIndex, size_t columnsCount = 1);
```

Метод `removeRow` позволяет удалить строки в количестве `rowCount` начиная с индекса `rowIndex`. Индексация строк начинается с нуля.

```
void removeRow(size_t rowIndex, size_t rowCount = 1);
```

Метод `setColumnsVisible` позволяет задавать видимость столбцов в количестве `columnsCount`, начиная с индекса `first`. Индексация столбцов начинается с нуля.

```
void setColumnsVisible(size_t first, size_t columnsCount, bool visible);
```

Метод `setRowsVisible` позволяет задавать видимость строк в количестве `rowCount`, начиная с индекса `first`. Индексация строк начинается с нуля.

```
void setRowsVisible(size_t first, size_t rowCount, bool visible);
```

Следующий набор методов позволяет группировать строки и колонки таблицы. Редактор дает возможность отображать группы в виде иерархии. Совместно с данными методами можно использовать методы `setColumnsVisible` и `setRowsVisible` чтобы раскрывать и закрывать фрагменты иерархии групп.

Методы могут вызвать исключения [OutOfRangeException](#) и [IncorrectArgumentError](#) в случае использования некорректных индексов, выходящих за рамки таблицы.

```
void groupRows(size_t first, size_t rowCount);
void ungroupRows(size_t first, size_t rowCount);
void clearRowGroups(size_t first, size_t rowCount);
void groupColumns(size_t first, size_t columnsCount);
void ungroupColumns(size_t first, size_t columnsCount);
void clearColumnGroups(size_t first, size_t columnsCount);
```

Метод `setColumnWidth` позволяет установить значение `width` в качестве ширины столбца с индексом `columnIndex`. Индексация столбцов начинается с нуля.

Метод вызывает исключение [OutOfRangeException](#), если параметр `columnIndex` выходит за границы таблицы.

```
void setColumnWidth(size_t columnIndex, float width);
```

Метод `setRowHeight` позволяет установить значение `height` в качестве высоты строки с индексом `rowIndex`. Индексация строк начинается с нуля.

Метод вызывает исключение [OutOfRangeException](#), если параметр `rowIndex` выходит за границы таблицы.

Класс `RowHeightRule` задает точность значения, где `Exact` – точно, а `AtLeast` – не меньше.

```
enum class RowHeightRule : std::uint8_t
{
    Exact,
    AtLeast
};

void setRowHeight(size_t rowIndex, float height,
                 boost::optional<RowHeightRule> heightRule);
```

4.25.2.4 Управление ячейками и диапазонами ячеек

Метод `getCell` позволяет получить доступ к ячейкам таблицы по адресу `cellRef` или положению, указанному в параметре `position`.

Параметр `cellRef` задает адрес ячейки с использованием наиболее распространенной системы адресации ячеек, при которой столбцы задаются буквами, а строки – числами, например, «A1».

Параметр `position` использует для адресации ячейки объект класса [CellPosition](#), который позволяет указать строки и столбцы по индексу. Например, положение ячейки «A1» задается как `CellPosition(0,0)`.

```
Cell getCell(const std::string& cellRef);  
Cell getCell(const CellPosition& position);
```

Метод `getCellRange` позволяет получить доступ к диапазону ячеек таблицы по адресу в параметре `range` или положению, указанному в параметре `CellRangePosition& range`.

Параметр `range` задает адрес диапазона ячеек с использованием наиболее распространенной системы адресации ячеек, при которой столбцы задаются буквами, а строки – числами, например, «A1:C4».

Параметр `CellRangePosition& range` использует для адресации диапазона ячеек объект класса [CellRangePosition](#), который позволяет указать строки и столбцы по индексу. Например, диапазон «A1:C4» задается как `CellRangePosition(0,0,2,3)`.

```
CellRange getCellRange(const std::string& range);  
CellRange getCellRange(const CellRangePosition& range);
```

4.25.2.5 Список диаграмм

Для получения списка диаграмм таблицы [Charts](#) используется метод `getCharts`.

```
Charts getCharts();
```

4.25.2.6 Список именованных выражений

Для получения списка именованных выражений [NamedExpressions](#) используется метод `getNamedExpressions`.

```
NamedExpressions getNamedExpressions();
```

4.25.2.7 Создание копии листа в табличном документе

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.

```
void duplicate();
```

4.25.2.8 Перемещение листа в табличном документе

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.

```
void moveTo(size_t index);
```

4.25.2.9 Управление видимостью листа в табличном документе

Для управления видимостью листа таблицы в табличном документе используется метод `setVisible`. Если значение параметра `visible` равно `true`, то лист таблицы отображается в редакторе таблиц.

```
void setVisible(bool visible);
```

Для проверки видимости листа таблицы в табличном документе используется метод `isVisible`.

```
bool isVisible() const;
```

4.25.2.10 Область печати в табличном документе

Для задания области печати в табличном документе для экспорта документа, печати и т.д. используется метод `setPrintArea`. Если значение параметра метода равно `boost::none`, то область печати считается неопределенной. Для задания множественных областей печати / экспорта используется метод `setPrintAreas`. Метод `getPrintAreas` возвращает коллекцию текущих областей печати.

```
void setPrintArea(const boost::optional<CellRangePosition>& range);  
void setPrintAreas(const CellRangePositions& ranges);  
CellRangePositions getPrintAreas() const;
```

4.25.2.11 Оператор ==

Оператор сравнения `==` используется для определения эквивалентности двух таблиц.

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

4.25.2.12 Оператор !=

Оператор сравнения `!=` используется для определения неэквивалентности двух таблиц.

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

4.25.3 Именованные выражения

Именованное выражение – это выражение (являющееся описанием диапазона или формулой), которому присвоено имя. Преимуществом именованного выражения является его информативность. Именованные выражения упрощают работу с ячейками, также их удобно использовать при работе с формулами. На данный момент доступен функционал для работы с именованными выражениями, представляющими собой ссылки на диапазоны ячеек.

4.25.3.1 Класс NamedExpression

Класс описывает структуру именованного выражения.

```
class NamedExpression
{
public:
    std::string getName() const;
    std::string getExpression() const;
    boost::optional<CellRange> getCellRange() const;
};
```

4.25.3.1.1 Метод getName

Возвращает имя именованного выражения.

```
std::string getName() const;
```

4.25.3.1.2 Метод getExpression

Возвращает текст выражения (формулы).

```
std::string getExpression() const;
```

4.25.3.1.3 Метод getCellRange

Возвращает диапазон ячеек [CellRange](#), если выражение является ссылкой на диапазон.

```
boost::optional<CellRange> getCellRange() const;
```

4.25.3.2 Класс NamedExpressions

Класс для представления списка именованных выражений.

```
class NamedExpressions
{
public:
    boost::optional<NamedExpression> get(const std::string& name);
    std::shared_ptr<Enumerator<NamedExpression>> getEnumerator();
};
```


4.25.3.2.1 Метод `get`

Возвращает именованное выражение [NamedExpression](#) по имени, если оно существует.

```
boost::optional<NamedExpression> get(const std::string& name);
```

4.25.3.2.2 Метод `getEnumerator`

Возвращает объект типа `Enumerator`, позволяющий получить доступ ко всему списку именованных выражений.

```
std::shared_ptr<Enumerator<NamedExpression>> getEnumerator();
```

4.25.3.3 Класс `NamedExpressionsValidationResult`

Представляет собой результат операции, связанной с именованным выражением.

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

Класс содержит следующие поля:

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

4.25.4 Сводные таблицы

4.25.4.1 Класс `PivotTable`

Класс для представления сводной таблицы.

```
class PivotTable
{
public:
    void remove();

    std::string getSourceRangeAddress() const;

    CellRange getSourceRange() const;
    CellRange getPivotRange() const;

    void changeSourceRange(const std::string& sourceRange);

    bool isRowGrandTotalEnabled() const;
    bool isColumnGrandTotalEnabled() const;

    PivotTableCaptions getPivotTableCaptions() const;

    PivotTableLayoutSettings getPivotTableLayoutSettings() const;
```

```

PivotTableUnsupportedFeatures getUnsupportedFeatures() const;

PivotTableFields getFieldsList() const;

PivotTableCategoryFields getRowFields() const;
PivotTableCategoryFields getColumnFields() const;

PivotTableValueFields getValueFields() const;

PivotTablePageFields getPageFields() const;

PivotTableFieldCategories getFieldCategories(const std::string& fieldName)
const;

PivotTableItems getFieldItems(const std::string& fieldName) const;

PivotTableItems getFieldItemsByName(const std::string& fieldName,
                                     const std::string& itemName) const;

boost::optional<PivotTableFilter> getFilter(const std::string& fieldName)
const;

PivotTableFilters getFilters() const;

PivotTableUpdateResult update();

PivotTableEditor createPivotTableEditor();
};

```

4.25.4.1.1 Метод remove

Метод удаляет сводную таблицу.

```
void remove();
```

4.25.4.1.2 Метод getSourceRangeAddress

Метод возвращает текстовое представление диапазона исходных данных сводной таблицы.

```
std::string getSourceRangeAddress() const;
```

4.25.4.1.3 Метод getSourceRange

Метод возвращает диапазон исходных данных сводной таблицы [CellRange](#).

```
CellRange getSourceRange() const;
```

4.25.4.1.4 Метод getPivotRange

Метод возвращает диапазон ячеек [CellRange](#), в котором размещена сводная таблица.

```
CellRange getPivotRange() const;
```

4.25.4.1.5 Метод `changeSourceRange`

Метод позволяет задать новый диапазон исходных данных сводной таблицы без обновления самой таблицы.

```
void changeSourceRange(const std::string& sourceRange);
```

4.25.4.1.6 Метод `isRowGrandTotalEnabled`

Метод возвращает `true`, если разрешено показывать общие итоги для строк.

```
bool isRowGrandTotalEnabled() const;
```

4.25.4.1.7 Метод `isColumnGrandTotalEnabled`

Метод возвращает `true`, если разрешено показывать общие итоги для столбцов.

```
bool isColumnGrandTotalEnabled() const;
```

4.25.4.1.8 Метод `getPivotTableCaptions`

Метод возвращает информацию о всех заголовках сводной таблицы [PivotTableCaptions](#).

```
PivotTableCaptions getPivotTableCaptions() const;
```

4.25.4.1.9 Метод `getPivotTableLayoutSettings`

Метод возвращает настройки отображения [PivotTableLayoutSettings](#) сводной таблицы.

```
PivotTableLayoutSettings getPivotTableLayoutSettings() const;
```

4.25.4.1.10 Метод `getUnsupportedFeatures`

Метод возвращает неподдерживаемые свойства [PivotTableUnsupportedFeatures](#) сводной таблицы.

```
PivotTableUnsupportedFeatures getUnsupportedFeatures() const;
```

4.25.4.1.11 Метод `getFieldsList`

Метод возвращает список всех полей сводной таблицы.

```
PivotTableFields getFieldsList() const;
```

Где `PivotTableFields` является коллекцией элементов типа [PivotTableField](#).

```
using PivotTableFields = std::vector<PivotTableField>;
```

4.25.4.1.12 Метод `getRowFields`

Метод возвращает список полей из области строк.

```
PivotTableCategoryFields getRowFields() const;
```

4.25.4.1.13 Метод getColumnFields

Метод возвращает список полей из области колонок.

```
PivotTableCategoryFields getColumnFields() const;
```

4.25.4.1.14 Метод getValueFields

Метод возвращает список полей из области значений.

```
PivotTableValueFields getValueFields() const;
```

4.25.4.1.15 Метод getPageFields

Метод возвращает список полей из области фильтров.

```
PivotTablePageFields getPageFields() const;
```

4.25.4.1.16 Метод getFieldCategories

Метод возвращает список категорий, содержащих заданное поле.

```
PivotTableFieldCategories getFieldCategories(const std::string& fieldName) const;
```

4.25.4.1.17 Метод getFieldItems

Метод возвращает все элементы сводной таблицы по заданному имени поля.

```
PivotTableItems getFieldItems(const std::string& fieldName) const;
```

4.25.4.1.18 Метод getFieldItemsByName

Метод возвращает все элементы из заданного поля по имени.

```
PivotTableItems getFieldItemsByName(const std::string& fieldName, const std::string& itemName) const;
```

4.25.4.1.19 Метод getFilter

Метод возвращает фильтр по заданному имени поля.

```
boost::optional<PivotTableFilter> getFilter(const std::string& fieldName) const;
```

4.25.4.1.20 Метод getFilters

Метод возвращает список фильтров сводной таблицы.

```
PivotTableFilters getFilters() const;
```

4.25.4.1.21 Метод update

Метод обновляет и полностью пересчитывает сводную таблицу.

```
PivotTableUpdateResult update();
```

4.25.4.1.22 Метод createPivotTableEditor

Метод возвращает объект [PivotTableEditor](#), который служит для обновления свойств и редактирования сводной таблицы.

```
PivotTableEditor createPivotTableEditor();
```

4.25.4.2 Структура PivotTableCaptions

Данная структура хранит все пользовательские заголовки сводной таблицы.

```
struct PivotTableCaptions
{
public:
    boost::optional<std::string> errorCaption;
    boost::optional<std::string> emptyCaption;

    boost::optional<std::string> grandTotalCaption;

    std::string valuesHeaderCaption;

    boost::optional<std::string> rowHeaderCaption;
    boost::optional<std::string> columnHeaderCaption;
};
```

Структура содержит следующие поля:

- `errorCaption` – алиас (псевдоним) для значений, которые возвращают ошибку;
- `emptyCaption` – алиас для значений, которые возвращают пустое значение;
- `grandTotalCaption` – алиас общих итогов;
- `valuesHeaderCaption` – алиас поля из области значений; это поле отображается в отчете в случае, если в сводной таблице наличие более двух полей из области значений, и макет имеет тип 'outline' или 'tabular';
- `rowHeaderCaption` – алиас заголовка строк (виден только при включенном компактном макете, это алиас по умолчанию);
- `columnHeaderCaption` – алиас заголовка колонок (виден только при включенном компактном макете, это алиас по умолчанию).

4.25.4.3 Структура PivotTableLayoutSettings

Структура содержит настройки отображения сводной таблицы.

```
struct PivotTableLayoutSettings
{
public:
    PivotTableReportLayout reportLayout;
    ValueFieldsOrientation valueFieldsOrientation;
    PageFieldOrder pageFieldOrder;
    IndentSize indentForCompactLayout;
    size_t pageFieldWrapCount;
    bool isMergeAndCenterLabelsEnabled;
    bool useGridDropZones;
    bool displayFieldCaptions;
};
```

Структура содержит следующие поля:

- reportLayout – настройка вида макета сводной таблицы [PivotTableReportLayout](#) (компактный, табличный, структурный);
- valueFieldsOrientation – позволяет расположить положение значений в случае, если в сводной таблице более двух полей значений ([ValueFieldsOrientation](#));
- pageFieldOrder – настройка порядка полей фильтров [PageFieldOrder](#) (вниз, затем поперек или сначала поперек, потом вниз);
- indentForCompactLayout – размер отступа [IndentSize](#) для полей в области строк в компактном макете (режим иерархии в случае наличия более двух полей);
- pageFieldWrapCount – настройка связана с полем pageFieldOrder, она показывает через сколько полей будет совершено указанное действие (перенос на следующую строку и т.д);
- isMergeAndCenterLabelsEnabled – настройка позволяет объединить ячейки заголовков;
- useGridDropZones – использовать классический вид (как в Excel 2003). Влияет только на расположение полей в отчете;
- displayFieldCaptions – отображать заголовки полей.

4.25.4.4 Класс PivotTableUnsupportedFeature

Перечисление, которое описывает неподдерживаемую функциональность.

```
enum class PivotTableUnsupportedFeature : uint8_t
{
    CalculatedField = 0,
    CalculatedItem,
    CollapsedValues,
    ShowDataAs,
};
```

Варианты неподдерживаемых свойств:

- CalculatedField – вычисляемые поля;
- CalculatedItem – вычисляемые элементы;
- CollapsedValues – свернутые элементы поля;
- ShowDataAs – дополнительные вычисления (Show data как в MS Excel).

4.25.4.5 Класс PivotTableReportLayout

Перечисление, описывающее внешний вид отчетов сводной таблицы.

```
enum class PivotTableReportLayout : uint8_t
{
    Compact = 0,
    Tabular,
    Outline,
};
```

Варианты вида отчетов:

- Compact – компактный;
- Tabular – табличный;
- Outline – структурный.

4.25.4.6 Класс ValueFieldsOrientation

Класс описывает варианты ориентации в случае, когда в сводной таблице более, чем одно поле из области значений.

```
enum class ValueFieldsOrientation : uint8_t
{
    ByRows = 0,
    ByColumns,
};
```

Варианты ориентации:

- ByRows – по строкам;
- ByColumns – по столбцам.

4.25.4.7 Класс PageFieldOrder

Класс описывает вид отображения полей из области фильтров.

```
enum class PageFieldOrder : uint8_t
{
    DownThenOver = 0,
    OverThenDown,
};
```

Варианты ориентации:

- DownThenOver – вниз, затем поперек;
- OverThenDown – поперек, затем вниз.

4.25.4.8 Класс PivotTableFieldCategory

Класс описывает флаги, которые задают категорию области полей.

```
enum class PivotTableFieldCategory : uint8_t
{
    Pages = 0,
    Rows,
    Columns,
    Values,
};
```

Поля класса соответствуют следующим областям:

- Pages – область фильтров;
- Rows – область строк;
- Columns – область колонок;
- Values – область значений.

4.25.4.9 Класс PivotTableFieldCategories

Класс обеспечивает доступ к списку категорий полей сводных таблиц.

```
class PivotTableFieldCategories
{
public:
    std::shared_ptr<Enumerator<PivotTableFieldCategory>> GetEnumerator();
};
```

4.25.4.9.1 Метод GetEnumerator

Возвращает объект Enumerator для доступа к коллекции категорий.

```
std::shared_ptr<Enumerator<PivotTableFieldCategory>> GetEnumerator();
```

4.25.4.10 Класс PivotTableFunction

Данное перечисление описывает функции, которые могут быть использованы в сводных таблицах.

```
using PivotTableFunctions = std::vector<PivotTableFunction>;

enum class PivotTableFunction : std::uint8_t
{
    Auto,
    Sum,
    Count,
    CountNums,
    Average,
    Max,
    Min,
    Product,
    StdDeviation,
    StdDeviationPopulation,
```



```
Variance,  
VariancePopulation  
};
```

Поля класса соответствуют следующим функциям:

- `Auto` – автозаполнение;
- `Sum` – суммирует все числовые данные;
- `Count` – количество всех ячеек;
- `CountNums` – количество числовых ячеек;
- `Average` – среднее значение;
- `Max` – наибольшее значение;
- `Min` – наименьшее значение;
- `Product` – произведение всех ячеек;
- `StdDeviation` – стандартное смещенное отклонение;
- `StdDeviationPopulation` – стандартное несмещенное отклонение;
- `Variance` – смещенная дисперсия;
- `VariancePopulation` – несмещенная дисперсия.

4.25.4.11 Класс `PivotTableFilter`

Представляет собой интерфейс для доступа к списку фильтров таблицы, каждый из которых обладает свойством видимости. При любом изменении фильтров они должны быть применены к сводной таблице посредством использования метода `setFilter` класса [PivotTableEditor](#).

```
class PivotTableFilter  
{  
public:  
    std::string getFieldname() const;  
  
    size_t getCount() const;  
  
    std::string getName(size_t itemIndex) const;  
  
    bool isHidden(size_t itemIndex) const;  
    void setHidden(size_t itemIndex, bool hidden);  
  
    bool isHidden(const std::string& itemName) const;  
    void setHidden(const std::string& itemName, bool hidden);  
};
```

4.25.4.11.1 Метод `getFieldName`

Возвращает имя поля, с которым ассоциирован фильтр.

```
std::string getFieldName() const;
```

4.25.4.11.2 Метод `getCount`

Возвращает количество фильтруемых полей.

```
size_t getCount() const;
```

4.25.4.11.3 Метод `getName`

Возвращает имя поля для заданного индекса.

```
std::string getName(size_t itemIndex) const;
```

4.25.4.11.4 Метод `isHidden`

Возвращает видимость поля для заданного индекса, если `true`, то поле скрыто.

```
bool isHidden(size_t itemIndex) const;
```

4.25.4.11.5 Метод `setHidden`

Устанавливает видимость поля для заданного индекса. Параметры: `itemName` – имя поля, `hidden` – видимость (`true` – поле скрыто).

```
void setHidden(const std::string& itemName, bool hidden);
```

4.25.4.12 Класс `PivotTableFilters`

Класс обеспечивает доступ к списку фильтров.

```
class PivotTableFilters
{
public:
    std::shared_ptr<Enumerator<PivotTableFilter>> getEnumerator();
};
```

4.25.4.12.1 Метод `getEnumerator`

Возвращает объект `Enumerator` для доступа к коллекции фильтров.

```
std::shared_ptr<Enumerator<PivotTableFilter>> getEnumerator();
```

4.25.4.13 Структура `PivotTableFieldProperties`

Структура содержит основные свойства полей сводной таблицы.

```
struct PivotTableFieldProperties
{
public:
    std::string fieldName;
    boost::optional<std::string> fieldAlias;
    boost::optional<std::string> subtotalAlias;
};
```

Структура содержит следующие поля:

- `fieldName` – имя поля;
- `fieldAlias` – псевдоним поля (пользовательское имя поля);
- `subtotalAlias` – псевдоним подытогов конкретного поля.

4.25.4.14 Структура `PivotTableField`

Структура содержит свойства полей сводной таблицы.

```
struct PivotTableField
{
public:
    PivotTableFieldProperties fieldProperties;
    PivotTableFieldCategories fieldCategories;

    boost::optional<std::string> customFormula;
};
```

Структура содержит следующие поля:

- `fieldProperties` – свойства полей сводной таблицы;
- `fieldCategories` – области полей сводной таблицы;
- `customFormula` – вычисляемая формула.

4.25.4.15 Структура `PivotTableCategoryField`

Структура, содержащая свойства поля сводной таблицы, использующегося как строка/столбец.

```
struct PivotTableCategoryField
{
    PivotTableFieldProperties fieldProperties;
    PivotTableFunctions subtotalFunctions;
};
```

Структура содержит следующие поля:

- `fieldProperties` – свойства поля;
- `subtotalFunctions` – список функций для вычисления подытога.

4.25.4.16 Структура `PivotTableValueField`

Структура, содержащая свойства поля сводной таблицы, использующегося как значение.

```
struct PivotTableValueField
{
    std::string baseFieldName;
    std::string valueFieldName;
};
```

```
CellFormat cellNumberFormat;  
PivotTableFunction totalFunction;  
  
boost::optional<std::string> customFormula;  
};
```

Структура содержит следующие поля:

- `baseFieldName` – оригинальное поле на основе которого было создано данное поле;
- `valueFieldName` – автоматический уникальный псевдоним такой как “Sum of % имя поля%”;
- `cellNumberFormat` – числовой формат для конкретного поля значений;
- `totalFunction` – агрегирующая функция поля значений (SUM, COUNT, MAX и т.д.);
- `customFormula` – вычисляемая формула для поля значений.

4.25.4.17 Структура `PivotTablePageField`

Структура, содержащая свойства поля из области фильтров.

```
struct PivotTablePageField  
{  
    PivotTableFieldProperties fieldProperties;  
};
```

4.25.4.18 Класс `PivotTableItem`

Класс описывает элемент сводной таблицы.

```
class PivotTableItem  
{  
public:  
    std::string getName() const;  
    boost::optional<std::string> getAlias() const;  
    PivotTableItemType getItemType() const;  
  
    bool isCollapsed() const;  
};
```

4.25.4.18.1 Метод `getName`

Метод возвращает имя элемента сводной таблицы.

```
std::string getName() const;
```

4.25.4.18.2 Метод `getAlias`

Метод возвращает псевдоним элемента (идентификатор, созданный пользователем).

```
boost::optional<std::string> getAlias() const;
```

4.25.4.18.3 Метод `getItemType`

Метод возвращает тип элемента сводной таблицы.

```
PivotTableItemType getItemType() const;
```

4.25.4.18.4 Метод `isCollapsed`

Метод возвращает `true`, если элемент сводной таблицы свернут.

```
bool isCollapsed() const;
```

4.25.4.19 Класс `PivotTableItemType`

Класс содержит возможные типы элементов сводной таблицы.

```
enum class PivotTableItemType : uint8_t
{
    Number,
    String,
    Boolean,
    DateTime,
    Empty,
    Error,

    NumberGroup,
    DateIntervalGroup,
    DateTimeGroup,
    CustomGroup
};
```

Значения полей класса определяют типы полей:

- `Number` – числовой;
- `String` – строковый;
- `Boolean` – логический;
- `DateTime` – дата / время;
- `Empty` – пустой тип;
- `Error` – ошибка;
- `NumberGroup` – интервальная группировка;
- `DateIntervalGroup` – интервальная группировка по датам;
- `DateTimeGroup` – группировка по дате / времени;
- `CustomGroup` – пользовательская (произвольная) группировка.

4.25.4.20 Класс `PivotTableEditor`

```
class PivotTableEditor
{
public:
    PivotTableEditor addField(const std::string& fieldName,
```

```

        PivotTableFieldCategory toCategory,
        boost::optional<size_t> index = boost::none);

PivotTableEditor moveField(const std::string& fieldName,
        PivotTableFieldCategory toCategory,
        boost::optional<size_t> index = boost::none);

PivotTableEditor removeField(const std::string& fieldName,
        PivotTableFieldCategory fromCategory);

PivotTableEditor reorderField(const std::string& fieldName,
PivotTableFieldCategory category, size_t toIndex);

PivotTableEditor enableField(const std::string& fieldName);

PivotTableEditor disableField(const std::string& fieldName);

PivotTableEditor setSummarizeFunction(const std::string& valueFieldName,
        PivotTableFunction summarizeFunction);

PivotTableEditor setFilter(const PivotTableFilter& filter);

PivotTableEditor setFilters(const PivotTableFilters& filters);

PivotTableEditor setCaptions(const PivotTableCaptions& captions);

PivotTableEditor setLayoutSettings(const PivotTableLayoutSettings&
layoutSettings);

PivotTableEditor setGrandTotalSettings(bool isRowGrandTotalEnabled,
        bool isColGrandTotalEnabled);

PivotTableUpdateResult apply();
};

```

4.25.4.20.1 Метод addField

Метод добавляет новое поле в сводную таблицу, используя параметры: `fieldName` - имя поля, `toCategory` - область поля, `index` - позиция в области.

```

PivotTableEditor addField(const std::string& fieldName,
        PivotTableFieldCategory toCategory,
        boost::optional<size_t> index = boost::none);

```

4.25.4.20.2 Метод moveField

Метод перемещает поле между областями. Параметры: `fieldName` - имя поля, `toCategory` - область, в которую перемещается поле, `index` - позиция в новой области.

```

PivotTableEditor moveField(const std::string& fieldName,
        PivotTableFieldCategory toCategory,
        boost::optional<size_t> index = boost::none);

```

4.25.4.20.3 Метод `removeField`

Метод удаляет поле из области. Параметры: `fieldName` - имя поля, `fromCategory` - область, из которой удаляется поле.

```
PivotTableEditor removeField(const std::string& fieldName,  
                             PivotTableFieldCategory fromCategory);
```

4.25.4.20.4 Метод `reorderField`

Метод изменяет позицию поля в пределах области. Параметры: `fieldName` - имя поля, `category` - область, `toIndex` - новая позиция поля.

```
PivotTableEditor reorderField(const std::string& fieldName,  
                              PivotTableFieldCategory category,  
                              size_t toIndex);
```

4.25.4.20.5 Метод `enableField`

Метод добавляет поле в область, зависящую от типа поля. Параметр `fieldName` - имя поля.

```
PivotTableEditor enableField(const std::string& fieldName);
```

4.25.4.20.6 Метод `disableField`

Метод удаляет поле из всех областей. Параметр `fieldName` - имя поля.

```
PivotTableEditor disableField(const std::string& fieldName);
```

4.25.4.20.7 Метод `setSummarizeFunction`

Метод задает суммирующую функцию для поля из области значений. Параметр `valueFieldName` - имя поля, `summarizeFunction` - суммирующая функция.

```
PivotTableEditor setSummarizeFunction(const std::string& valueFieldName,  
                                       PivotTableFunction summarizeFunction);
```

4.25.4.20.8 Метод `setFilter`

Метод задает фильтр сводной таблицы. Если фильтр не может быть применен, вызывается исключение `PivotTableError`.

```
PivotTableEditor setFilter(const PivotTableFilter& filter);
```

4.25.4.20.9 Метод `setFilters`

Метод задает фильтры сводной таблицы. Если какой-то из фильтров не может быть применен, он пропускается.

```
PivotTableEditor setFilters(const PivotTableFilters& filters);
```

4.25.4.20.10 Метод `setCaptions`

Метод задает заголовки сводной таблицы.

```
PivotTableEditor setCaptions(const PivotTableCaptions& captions);
```

4.25.4.20.11 Метод `setLayoutSettings`

Метод устанавливает настройки отображения [PivotTableLayoutSettings](#) сводной таблицы.

```
PivotTableEditor setLayoutSettings(const PivotTableLayoutSettings& layoutSettings);
```

4.25.4.20.12 Метод `setGrandTotalSettings`

Метод задает настройки отображения общего итога. Параметры: `isRowGrandTotalEnabled` – показывать общие итоги для строк, `isColGrandTotalEnabled` – показывать общие итоги для столбцов.

```
PivotTableEditor setGrandTotalSettings(bool isRowGrandTotalEnabled, bool isColGrandTotalEnabled);
```

4.25.4.20.13 Метод `apply`

Метод обновляет сводную таблицу с заданными свойствами.

```
PivotTableUpdateResult apply();
```

4.25.4.21 Класс `PivotTableUpdateResult`

Данное перечисление описывает возможные результаты обновления сводной таблицы.

```
enum class PivotTableUpdateResult : uint8_t
{
    Success,
    NoPivotTable,
    NoSuchFieldInCategory,
    NoSuchFieldInPivotTable,
    InvalidIndex,
    FieldAlreadyEnabled,
    MovingFieldToTheSameCategoryForbidden,
    InvalidFunction,
    InvalidDataSourceRange,
    NoDataRowsInDataSource,
    EmptyDataSourceHeaders,
    NoReferenceUnderDefine,
    AnotherPivotInsideDataSource
};
```

Значения полей класса:

- `Success` – успешное обновление таблицы;
- `NoPivotTable` – сводная таблица не найдена;

- NoSuchFieldInCategory – не найдено поле в области;
- NoSuchFieldInPivotTable – не найдено поле в сводной таблице;
- InvalidIndex – ошибка в индексе;
- FieldAlreadyEnabled – поле уже разрешено;
- MovingFieldToTheSameCategoryForbidden – попытка перемещения поля в рамках текущей области;
- InvalidFunction – неправильная функция;
- InvalidDataSourceRange – ошибка диапазона исходных данных;
- NoDataRowsInDataSource – в исходных данных нет строк с данными;
- EmptyDataSourceHeaders – пустые заголовки исходных данных;
- NoReferenceUnderDefine – попытка обновить/создать сводную таблицу на именованном диапазоне, который не содержит ссылку, а содержит константу;
- AnotherPivotInsideDataSource – найдена другая сводная таблица в этом же диапазоне.

4.25.4.22 Класс PivotTablesManager

```
class PivotTablesManager
{
public:
    PivotTable create(CellRange cellRange,
                     boost::optional<Cell> destination = boost::none);

    PivotTable create(std::string formula,
                     boost::optional<Cell> destination = boost::none);
};
```

4.25.4.22.1 Метод create

Метод создает сводную таблицу на основе диапазона исходных данных. Если местоположение (*destination*) не задано, создается новый лист (таблица), и сводная таблица будет расположена по умолчанию.

```
PivotTable create(CellRange cellRange,
                  boost::optional<Cell> destination = boost::none);
```

Метод создает сводную таблицу на основе строки формулы. Если местоположение (*destination*) не задано, создается новая таблица и сводная таблица будет расположена по умолчанию.

```
PivotTable create(std::string formula,
                  boost::optional<Cell> destination = boost::none);
```

4.25.5 Диаграммы

4.25.5.1 Класс Chart

Класс представляет диаграмму в документе со всеми элементами (заголовок, легенда, тип, данные, диапазон и т.д.).

```
class Chart
{
public:
    ChartType getType() const;
    void setType(ChartType chartType);

    std::size_t getRangesCount() const;

    ChartRangeInfo getRange(std::size_t rangesIndex) const;

    boost::optional<std::string> getTitle() const;

    void setRange(const CellRangePosition& chartRange);

    void setRect(const Rect<float>& rect);

    bool isEmpty() const;

    bool isSolidRange() const;

    bool is3D() const;

    ChartSeriesDirectionType getDirectionType() const;

    ChartLabelsInfo getChartLabels() const;

    std::string getRangeAsString() const;

    void applySettings(const boost::optional<TableRangeInfo>& rangeInfo,
                      boost::optional<ChartSeriesDirectionType> directionType,
                      const boost::optional<std::string>& title,
                      boost::optional<ChartLabelsInfo> labelsInfo);
};
```

4.25.5.1.1 Метод getType

Метод возвращает тип диаграммы [ChartType](#).

```
ChartType getType() const;
```

4.25.5.1.2 Метод setType

Метод устанавливает тип диаграммы [ChartType](#). Параметр chartType - новый тип диаграммы.

```
void setType(ChartType chartType);
```

4.25.5.1.3 Метод `getRangesCount`

Метод возвращает количество серий диаграммы.

```
std::size_t getRangesCount() const;
```

4.25.5.1.4 Метод `getRange`

Метод возвращает диапазон ячеек с исходными данными диаграммы.

```
ChartRangeInfo getRange(std::size_t rangesIndex) const;
```

4.25.5.1.5 Метод `getTitle`

Метод возвращает заголовок диаграммы [ChartType](#).

```
boost::optional<std::string> getTitle() const;
```

4.25.5.1.6 Метод `setRange`

Метод задает диапазон ячеек с исходными данными для диаграммы.

```
void setRange(const CellRangePosition& chartRange);
```

4.25.5.1.7 Метод `setRect`

Метод `setRect` задает область расположения диаграммы.

```
void setRect(const Rect<float>& rect);
```

4.25.5.1.8 Метод `isEmpty`

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

```
bool isEmpty() const;
```

4.25.5.1.9 Метод `isSolidRange`

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

```
bool isSolidRange() const;
```

4.25.5.1.10 Метод `is3D`

Метод возвращает `true`, если диаграмма трехмерная.

```
bool is3D() const;
```

4.25.5.1.11 Метод `getDirectionType`

Метод возвращает направление серий диаграммы.

```
ChartSeriesDirectionType getDirectionType() const;
```

4.25.5.1.12 Метод `getChartLabels`

Метод возвращает коллекцию меток диаграммы типа [ChartLabelsInfo](#).

```
ChartLabelsInfo getChartLabels() const;
```

4.25.5.1.13 Метод `getRangeAsString`

Метод возвращает диапазон ячеек диаграммы в формате строки.

```
std::string getRangeAsString() const;
```

4.25.5.1.14 Метод `applySettings`

Метод позволяет применить следующие параметры к текущей выбранной диаграмме: `rangeInfo` – обновленный диапазон исходных данных диаграммы, `directionType` – направление серий, `title` – заголовок диаграммы, `labelsInfo` – информация о метках диаграммы.

```
void applySettings(const boost::optional<TableRangeInfo>& rangeInfo,  
                 boost::optional<ChartSeriesDirectionType> directionType,  
                 const boost::optional<std::string>& title,  
                 boost::optional<ChartLabelsInfo> labelsInfo);
```

4.25.5.2 Класс `Charts`

Класс `Charts` предоставляет интерфейс для доступа к списку диаграмм документа.

```
class Charts  
{  
public:  
    std::size_t getChartsCount() const;  
    Chart getChart(std::size_t chartIndex);  
    boost::optional<size_t> getChartIndexByDrawingIndex(std::size_t  
drawingIndex);  
};
```

4.25.5.2.1 Метод `getChartsCount`

Метод возвращает общее количество диаграмм.

```
std::size_t getChartsCount() const;
```

4.25.5.2.2 Метод `getChart`

Метод возвращает диаграмму [Chart](#) по индексу в коллекции диаграмм.

```
Chart getChart(std::size_t chartIndex);
```

4.25.5.2.3 Метод `getChartIndexByDrawingIndex`

Метод возвращает диаграмму [Chart](#) по индексу отрисовки `drawingIndex`.

```
boost::optional<size_t> getChartIndexByDrawingIndex(std::size_t drawingIndex);
```

4.25.5.3 Класс `ChartLabelsDetectionMode`

Класс описывает режимы автоматического определения меток диаграмм.

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

Поля класса соответствуют следующим режимам автоматического определения меток диаграмм:

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

4.25.5.4 Структура `ChartLabelsInfo`

Структура описывает настройки автоматического определения меток диаграммы.

Инициализируется конструктором, в который передаются параметры: `categoriesMode` – режим автоматического определения меток для областей, `seriesNameMode` – режим автоматического определения меток для серий, `oneColumnRow` – передается `true`, если диапазон диаграммы содержит только одну строку или одну колонку.

```
struct ChartLabelsInfo
{
    ChartLabelsInfo(const ChartLabelsDetectionMode categoriesMode,
                   const ChartLabelsDetectionMode seriesNameMode,
                   const bool oneColumnRow);

    ChartLabelsDetectionMode categoriesMode;
    ChartLabelsDetectionMode seriesNameMode;
    bool isOneColumnRowChart = false;
};
```

Поля структуры:

- `categoriesMode` – режим автоматического определения меток для областей [ChartLabelsDetectionMode](#);
- `seriesNameMode` – режим автоматического определения меток для серий [ChartLabelsDetectionMode](#);
- `isOneColumnRowChart` – поле содержит `true`, если диапазон диаграммы содержит только одну строку или одну колонку.

4.25.5.5 Структура `ChartRangeInfo`

Структура описывает серию диаграммы. Инициализируется конструктором, в который передаются следующие параметры: `tableRangeInfo` – диапазон ячеек, `color` – цвет серии диаграммы, `hidden` – видимость серии, `rangeType` – тип диапазона исходных данных диаграммы.

```
struct ChartRangeInfo
{
    ChartRangeInfo(const TableRangeInfo& tableRangeInfo,
                  const ColorRGBA& color,
                  const bool hidden,
                  const ChartRangeType rangeType);

    TableRangeInfo tableRangeInfo;
    ColorRGBA rangeColor;
    bool isHidden = false;
    ChartRangeType rangeType;
};
```

Поля структуры:

- `tableRangeInfo` – исходный диапазон ячеек для серии;
- `rangeColor` – цвет для отрисовки серии;
- `isHidden` – задает видимость серии диаграммы;
- `rangeType` – тип диапазона диаграммы.

4.25.5.6 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.

4.25.5.7 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.

4.25.5.8 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 соответствуют 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.

4.25.6 Структура `TableRangeInfo`

Предоставляет положение диапазона ячеек в таблице.

```
struct TableRangeInfo
{
    TableRangeInfo(const CellRangePosition& range): tableRange(range)
    {
    }

    CellRangePosition tableRange;
};
```

4.25.6.1 Конструктор `TableRangeInfo`

Является конструктором по умолчанию.

```
TableRangeInfo(const CellRangePosition& range) : tableRange(range)
```

4.25.6.2 Поля структуры `TableRangeInfo`

Структура `TableRangeInfo` содержит поле:

- tableRange – диапазон [TableRangeInfo](#), содержащий ячейки.

4.25.7 Класс Paragraphs

Класс Paragraphs предоставляет интерфейс для доступа к абзацам документа.

```
class Paragraphs
{
public:
    void setListSchema(boost::optional<ListSchema> schema);

    void setListLevel(boost::optional<size_t> level);

    void increaseListLevel();

    void decreaseListLevel();

    std::shared_ptr<Enumerator<Paragraph>> getEnumerator();
};
```

4.25.7.1 Метод setListSchema

Метод setListSchema позволяет установить тип маркированного или нумерованного списка (см. описание класса [ListSchema](#)).

```
void setListSchema(boost::optional<ListSchema> schema);
```

4.25.7.2 Методы setListLevel, increaseListLevel, decreaseListLevel

Методы позволяют управлять глубиной вложенности элемента списка. Значение может быть равным boost::none, если схема нумерации не установлена для абзаца. В этом случае будет установлено минимальное значение.

```
void setListLevel(boost::optional<size_t> level);
```

Увеличивает уровень списка на единицу. В случае, если максимальный уровень уже установлен, увеличения не происходит.

```
void increaseListLevel();
```

Уменьшает уровень списка на единицу. В случае, если минимальный уровень уже установлен, уменьшения не происходит.

```
void decreaseListLevel();
```

4.25.7.3 Метод getEnumerator

Возвращает объект Enumerator для доступа к коллекции абзацев.

```
std::shared_ptr<Enumerator<Paragraph>> getEnumerator();
```

4.25.8 Класс Paragraph

Класс Paragraph в объектной модели документа представляет отдельный абзац текста и является наследником класса Block.

```
class Paragraph
{
public:
    ParagraphProperties getParagraphProperties();
    void setParagraphProperties(const ParagraphProperties& properties);
    boost::optional<ListSchema> getListSchema();
    void setListSchema(boost::optional<ListSchema> schema);

    boost::optional<size_t> getListLevel();
    void setListLevel(boost::optional<size_t> level);

    void increaseListLevel();
    void decreaseListLevel();
};
```

4.25.8.1 Методы getParagraphProperties, setParagraphProperties

Методы позволяют управлять свойствами абзаца ([ParagraphProperties](#): выравнивание, отступ, интервалы). Свойства, имеющие значение `boost::none`, не будут изменяться.

```
ParagraphProperties getParagraphProperties();
void setParagraphProperties(const ParagraphProperties& properties);
```

Для управления настройками цвета текста и цвета выделения абзаца в ячейке используется объект класса [TextProperties](#).

```
TextProperties textProperties;
textProperties.textColor = ColorRGBA(255, 0, 0, 0);
cell->getRange().setTextProperties(textProperties);
```

4.25.8.2 Методы getListSchema, setListSchema

Методы позволяют управлять схемой форматирования списка, которая может быть применена к абзацам текста.

Метод `setListSchema` позволяет установить тип маркированного или нумерованного списка (см. описание класса [ListSchema](#)).

Метод `getListSchema` возвращает схему форматирования, либо значение `boost::none`, если схема нумерации не установлена для абзаца.

```
boost::optional<ListSchema> getListSchema();
void setListSchema(boost::optional<ListSchema> schema);
```

4.25.8.3 Методы `getListLevel`, `setListLevel`, `increaseListLevel`, `decreaseListLevel`

Методы позволяют управлять глубиной вложенности элемента списка. Значение может быть равным `boost::none`, если схема нумерации не установлена для абзаца. В этом случае будет установлено минимальное значение.

```
boost::optional<size_t> getListLevel();  
void setListLevel(boost::optional<size_t> level);
```

Увеличивает уровень списка на единицу. В случае, если максимальный уровень уже установлен, увеличения не происходит.

```
void increaseListLevel();
```

Уменьшает уровень списка на единицу. В случае, если минимальный уровень уже установлен, уменьшения не происходит.

```
void decreaseListLevel();
```

4.25.9 Структура `ParagraphProperties`

Структура `ParagraphProperties` представляет свойства абзаца.

```
using SpacingSize = float;  
using IndentSize = float;  
  
struct ParagraphProperties  
{  
    boost::optional<SpacingSize> beforeSpacing;  
    boost::optional<SpacingSize> afterSpacing;  
    boost::optional<LineSpacing> lineSpacing;  
    boost::optional<Alignment> alignment;  
    boost::optional<IndentSize> firstLineIndent;  
    boost::optional<IndentSize> leftIndent;  
    boost::optional<IndentSize> rightIndent;  
  
    bool::operator==(const ParagraphProperties& other) const;  
    bool::operator!=(const ParagraphProperties& other) const;  
};
```

Структура `ParagraphProperties` содержит следующие поля:

- `beforeSpacing` – интервал перед абзацем;
- `afterSpacing` – интервал после абзаца;
- `lineSpacing` – межстрочный интервал;
- `alignment` – выравнивание абзаца;
- `firstLineIndent` – отступ первой строки;
- `leftIndent` – отступ слева;
- `rightIndent` – отступ справа.

4.25.9.1 Оператор ==

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

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

4.25.9.2 Оператор !=

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

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

4.25.10 Класс Shape

Класс Shape описывает фигуру, содержит методы для установки и получения свойств [ShapeProperties](#).

```
class Shape : public Block
{
public:
    ShapeProperties getShapeProperties() const;
    void setShapeProperties(const ShapeProperties& properties);
};
```

4.25.11 Класс ShapeProperties

Класс описывает свойства фигуры и содержит следующие поля: [verticalAlignment](#) – вертикальное выравнивание, [borderProperties](#) – свойства границ фигуры, [fill](#) – свойства заполнения фигуры, [shapeTextLayout](#) – свойства текста внутри фигуры.

```
struct ShapeProperties
{
    boost::optional<VerticalAlignment> verticalAlignment;
    boost::optional<LineProperties> borderProperties;
    boost::optional<Fill> fill;
    boost::optional<ShapeTextLayout> shapeTextLayout;
};
```

4.25.12 Класс ShapeTextLayout

Класс описывает свойства текста, находящегося в фигуре.

```
enum class ShapeTextLayout : uint8_t
{
    DoNotAutoFit = 0,
    FitShapeExtentToText,
    FitTextToShape,
};
```

Поля класса соответствуют следующим значениям:

- DoNotAutoFit – размещение текста внутри фигуры по умолчанию;
- FitShapeExtensionToText – расширение фигуры под размер текста;
- FitTextToShape – заполнение фигуры текстом.

4.25.13 Класс Field

Класс Field наследуется от класса Block и предназначен для реализации некоторых полей, например, таких как содержание.

```
class Field : public Block
{
};
```

4.25.14 Класс Scripts

Класс Scripts предоставляет интерфейс для доступа к коллекции макрокоманд в документе.

```
class Scripts
{
public:
    virtual std::shared_ptr<Script> getScript(const std::string& name) const = 0;

    virtual void setScript(const std::string& name, const std::string& script) =
0;
    virtual void removeScript(const std::string& name) = 0;
    virtual std::shared_ptr<Enumerator<std::shared_ptr<Script>>> getEnumerator()
= 0;
};
```

4.25.14.1 Метод getScript

Возвращает объект для управления макрокомандой с названием name. Если макрокоманда с указанным названием не существует, то возвращает nullptr.

```
virtual std::shared_ptr<Script> getScript(const std::string& name) const = 0;
```

4.25.14.2 Метод setScript

Добавляет в коллекцию новую макрокоманду с текстом script и названием name. Если макрокоманда с таким названием уже существует, то ее текст будет заменен.

```
virtual void setScript(const std::string& name, const std::string& script) = 0;
```

4.25.14.3 Метод removeScript

Удаляет макрокоманду с названием name.

```
virtual void removeScript(const std::string& name) = 0;
```

4.25.14.4 Метод `GetEnumerator`

Возвращает объект типа `Enumerator` для перебора коллекции макрокоманд.

```
virtual std::shared_ptr<Enumerator<std::shared_ptr<Script>>> GetEnumerator() = 0;
```

4.25.15 Класс `Script`

Класс `Script` предоставляет интерфейс для доступа к отдельным макрокомандам документа.

```
class Script
{
public:
    virtual std::string getName() const = 0;
    virtual void setName(const std::string& name) = 0;
    virtual std::string getBody() const = 0;
    virtual void setBody(const std::string& body) = 0;
};
```

4.25.15.1 Метод `getName`

Возвращает название макрокоманды.

```
virtual std::string getName() const = 0;
```

4.25.15.2 Метод `setName`

Устанавливает название макрокоманды.

```
virtual void setName(const std::string& name) = 0;
```

4.25.15.3 Метод `getBody`

Возвращает текст исходного кода макрокоманды на языке программирования Lua.

```
virtual std::string getBody() const = 0;
```

4.25.15.4 Метод `setBody`

Устанавливает текст исходного кода макрокоманды на языке программирования Lua.

```
virtual void setBody(const std::string& body) = 0;
```

4.25.16 Класс `Blocks`

Класс `Blocks` предоставляет интерфейс для доступа к блокам документа.

```
class Blocks
{
public:
    boost::optional<Block> getBlock(std::size_t blockIndex);
    boost::optional<Paragraph> getParagraph(std::size_t paragraphIndex);
    boost::optional<Shape> getShape(std::size_t shapeIndex);
    boost::optional<Table> getTable(std::size_t tableIndex);
    boost::optional<Field> getField(std::size_t fieldIndex);
};
```

```
boost::optional<Table> getTable(const ID& tableID);

std::shared_ptr<Enumerator<Block>> getEnumerator();
std::shared_ptr<Enumerator<Paragraph>> getParagraphsEnumerator();
std::shared_ptr<Enumerator<Shape>> getShapesEnumerator();
std::shared_ptr<Enumerator<Table>> getTablesEnumerator();
std::shared_ptr<Enumerator<Field>> getFieldsEnumerator();
};
```

4.25.16.1 Методы `getBlock`, `getParagraph`, `getShape`, `getTable`, `getField`

Возвращают блок с соответствующим типом по индексу или `boost::none`, если блок с таким индексом не существует в документе.

```
boost::optional<Block> getBlock(std::size_t blockIndex);
boost::optional<Paragraph> getParagraph(std::size_t paragraphIndex);
boost::optional<Shape> getShape(std::size_t shapeIndex);
boost::optional<Table> getTable(std::size_t tableIndex);
boost::optional<Field> getField(std::size_t fieldIndex);
```

4.25.16.2 Методы `getEnumerator`, `getParagraphsEnumerator`, `getShapesEnumerator`, `getTablesEnumerator`, `getFieldsEnumerator`

Возвращают объект `Enumerator` для доступа к коллекции блоков соответствующего типа.

```
std::shared_ptr<Enumerator<Block>> getEnumerator();
std::shared_ptr<Enumerator<Paragraph>> getParagraphsEnumerator();
std::shared_ptr<Enumerator<Shape>> getShapesEnumerator();
std::shared_ptr<Enumerator<Table>> getTablesEnumerator();
std::shared_ptr<Enumerator<Field>> getFieldsEnumerator();
```

4.25.17 Класс `Block`

Класс `Block` является базовым классом для всех блоков документа.

```
class Block
{
public:
    boost::optional<Paragraph> toParagraph() const;
    boost::optional<Table> toTable() const;
    boost::optional<Shape> toShape() const;
    boost::optional<Field> toField() const;

    Range getRange();

    void remove();

    Section getSection();
};
```

4.25.17.1 Методы `toParagraph`, `toTable`, `toShape`, `toField`

Преобразует объект `Block` в блок соответствующего типа. Если это невозможно,

возвращает `boost::none`.

```
boost::optional<Paragraph> toParagraph() const;  
boost::optional<Table> toTable() const;  
boost::optional<Shape> toShape() const;  
boost::optional<Field> toField() const;
```

4.25.17.2 Метод `getRange`

Возвращает диапазон [Range](#), в котором содержится данный блок.

```
Range getRange();
```

4.25.17.3 Метод `remove`

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

```
void remove();
```

4.25.17.4 Метод `getSection`

Возвращает раздел, содержащий блок.

```
Section getSection();
```

4.25.18 Класс `Borders`

Класс `Borders` предоставляет методы для управления границами диапазона ячеек. Он позволяет установить внутренние границы другого стиля, отличающегося от внешних границ (слева, справа, сверху и снизу).

```
class Borders  
{  
public:  
    boost::optional<LineProperties> getLeft() const;  
    boost::optional<LineProperties> getRight() const;  
    boost::optional<LineProperties> getTop() const;  
    boost::optional<LineProperties> getBottom() const;  
    boost::optional<LineProperties> getDiagonalDown() const;  
    boost::optional<LineProperties> getDiagonalUp() const;  
    boost::optional<LineProperties> getInnerHorizontal() const;  
    boost::optional<LineProperties> getInnerVertical() const;  
  
    Borders& setLeft(boost::optional<LineProperties> lineProperties);  
    Borders& setRight(boost::optional<LineProperties> lineProperties);  
    Borders& setTop(boost::optional<LineProperties> lineProperties);  
    Borders& setBottom(boost::optional<LineProperties> lineProperties);  
    Borders& setDiagonalDown(boost::optional<LineProperties> lineProperties);  
    Borders& setDiagonalUp(boost::optional<LineProperties> lineProperties);  
  
    Borders& setOuter(boost::optional<LineProperties> lineProperties);  
    Borders& setDiagonals(boost::optional<LineProperties> lineProperties);  
  
    Borders& setInnerHorizontal(boost::optional<LineProperties> lineProperties);
```



```
Borders& setInnerVertical(boost::optional<LineProperties> lineProperties);
Borders& setInner(boost::optional<LineProperties> lineProperties);
Borders& setAll(boost::optional<LineProperties> lineProperties);
};
```

4.25.18.1 Методы для считывания свойств

Позволяет получить свойства границ [LineProperties](#). Свойства границ, имеющие значение `boost::none`, являются неопределенными.

```
boost::optional<LineProperties> getLeft() const;
boost::optional<LineProperties> getRight() const;
boost::optional<LineProperties> getTop() const;
boost::optional<LineProperties> getBottom() const;
boost::optional<LineProperties> getDiagonalDown() const;
boost::optional<LineProperties> getDiagonalUp() const;
boost::optional<LineProperties> getInnerHorizontal() const;
boost::optional<LineProperties> getInnerVertical() const;
```

4.25.18.2 Методы для установки свойств

Методы устанавливают свойства границы [LineProperties](#), возвращают тип [Borders](#), что удобно для формирования цепочки методов в одном выражении. Свойства границ, у которых свойства имеют значение `boost::none`, не будут изменены.

```
Borders& setLeft(boost::optional<LineProperties> lineProperties);
Borders& setRight(boost::optional<LineProperties> lineProperties);
Borders& setTop(boost::optional<LineProperties> lineProperties);
Borders& setBottom(boost::optional<LineProperties> lineProperties);
Borders& setDiagonalDown(boost::optional<LineProperties> lineProperties);
Borders& setDiagonalUp(boost::optional<LineProperties> lineProperties);
Borders& setOuter(boost::optional<LineProperties> lineProperties);
Borders& setDiagonals(boost::optional<LineProperties> lineProperties);
Borders& setInnerHorizontal(boost::optional<LineProperties> lineProperties);
Borders& setInnerVertical(boost::optional<LineProperties> lineProperties);
Borders& setInner(boost::optional<LineProperties> lineProperties);
Borders& setAll(boost::optional<LineProperties> lineProperties);
```

4.25.19 Класс RangeBorders

Класс `RangeBorders` оставлен для обратной совместимости. Вместо него следует использовать класс [Borders](#).

```
class RangeBorders public Borders
{
};
```

4.25.20 Структура `CellPosition`

Структура `CellPosition` представляет положение ячейки в таблице.

```
struct CellPosition
{
    CellPosition();
    CellPosition(const std::size_t rowArg, const std::size_t columnArg);

    std::string toString() const;

    std::size_t row;
    std::size_t column;
};
```

4.25.20.1 Метод `CellPosition`

Конструктор по умолчанию. Создается позиция соответствующая верхней левой ячейке 0,0 таблицы для редактора текста, либо ячейке A1 для редактора таблиц.

```
CellPosition();
```

Конструктор класса, в качестве параметров использует колонку `rowArg` и строку `columnArg`. Значение 0,0 соответствует верхней левой ячейке таблицы для редактора текста, либо ячейке A1 для редактора таблиц.

```
CellPosition(const std::size_t rowArg, const std::size_t columnArg);
```

4.25.20.2 Метод `toString`

Возвращает информацию о положении ячейки в виде строкового значения формата (row: <value>, column: <value>).

```
std::string toString() const;
```

4.25.20.3 Поля структуры `CellPosition`

Поля структуры `row` и `column` содержат значения строки и столбца, в которых находится ячейка.

4.25.21 Структура `CellRangePosition`, коллекция `CellRangePositions`

Структура `CellRangePosition` представляет положение диапазона ячеек в таблице. Коллекция `CellRangePositions` определена как `std::vector`, параметризованный типом `CellRangePosition`.

```
struct CellRangePosition
{
    CellRangePosition();
    CellRangePosition(const CellPosition& topLeftArg,
                     const CellPosition& bottomRightArg);
    CellRangePosition(const std::size_t topLeftRow,
```

```

        const std::size_t topLeftColumn,
        const std::size_t bottomRightRow,
        const std::size_t bottomRightColumn);

std::string toString() const;

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

CellPosition topLeft;
CellPosition bottomRight;
};

using CellRangePositions = std::vector<CellRangePosition>;

```

4.25.21.1 Метод CellRangePosition

Конструктор по умолчанию. Создается позиция соответствующая верхней левой ячейке 0,0 таблицы для редактора текста, либо ячейке A1 для редактора таблиц.

```
CellRangePosition();
```

Конструктор класса адресует прямоугольный диапазон, задаваемый позициями `topLeftArg` и `bottomRightArg`. Значение `topLeftArg=(0,0)` соответствует верхней левой ячейке таблицы для редактора текста, либо ячейке A1 для редактора таблиц.

```
CellRangePosition(const CellPosition& topLeftArg, const CellPosition&
bottomRightArg);
```

Конструктор класса адресует прямоугольный диапазон, задаваемый несколькими значениями.

```
CellRangePosition(const std::size_t topLeftRow,
                  const std::size_t topLeftColumn,
                  const std::size_t bottomRightRow,
                  const std::size_t bottomRightColumn);
```

4.25.21.2 Метод toString

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

```
std::string toString() const;
```

4.25.21.3 Поля структуры CellRangePosition

Поля `topLeft` и `bottomRight` содержат соответственно левую верхнюю и правую нижнюю позиции вершин прямоугольного диапазона ячеек в таблице.

4.25.22 Класс CellRange

Класс CellRange представляет диапазон (коллекцию) ячеек.

```
class CellRange
{
public:
    Table getTable();
    size_t getBeginRow() const;
    size_t getBeginColumn() const;
    size_t getLastRow() const;
    size_t getLastColumn() const;
    std::shared_ptr<Enumerator<Cell>> getEnumerator();
    void setBorders(const RangeBorders& borders);
    void insertCurrentDateTime(DateTimeFormat format);
    void setCellProperties(const CellProperties& cellProperties);
    CellProperties getCellProperties() const;
    void merge();
    bool autoFill(const& CellPosition destination);
};
```

4.25.22.1 Метод getTable

Возвращает таблицу, содержащую текущий диапазон.

```
Table getTable();
```

4.25.22.2 Метод getBeginRow

Возвращает индекс начальной строки диапазона.

```
size_t getBeginRow() const = 0;
```

4.25.22.3 Метод getBeginColumn

Возвращает индекс начального столбца диапазона.

```
size_t getBeginColumn() const;
```

4.25.22.4 Метод getLastRow

Возвращает индекс последней строки диапазона.

```
size_t getLastRow() const;
```

4.25.22.5 Метод getLastColumn

Возвращает индекс последнего столбца диапазона.

```
size_t getLastColumn() const;
```

4.25.22.6 Метод getEnumerator

Возвращает объект типа Enumerator для доступа к коллекции ячеек.

```
std::shared_ptr<Enumerator<Cell>> getEnumerator();
```

4.25.22.7 Метод `setBorders`

Устанавливает границы ячеек [RangeBorders](#) в диапазоне.

```
void setBorders(const RangeBorders& borders);
```

4.25.22.8 Метод `setCellProperties`

Служит для установки свойств ячеек диапазона [CellProperties](#).

```
void setCellProperties(const CellProperties& cellProperties);
```

4.25.22.9 Метод `getCellProperties`

Позволяет получить свойства ячеек диапазона [CellProperties](#). Возвращает структуру с одинаковыми свойствами для всех ячеек в диапазоне.

```
CellProperties getCellProperties() const;
```

4.25.22.10 Метод `autoFill`

Метод заполняет диапазон ячеек, используя данные из этого диапазона в качестве источника. Конечный диапазон вычисляется из начальной позиции исходного диапазона и последней позиции (аргумент `destination`). Таким образом, конечный диапазон всегда полностью содержит исходный диапазон. Метод `autoFill` автоматически интерполирует исходные точки и находит алгоритм аппроксимации, который используется для экстраполяции значений в диапазоне ячеек назначения. Результат выполнения метода в текстовом редакторе может отличаться от табличного редактора из-за разных типов данных в ячейках.

Метод возвращает `true`, если ячейки успешно заполнены и `false` в других случаях (например, если диапазон ячеек назначения содержит формулу или сводную таблицу и т. д.). Вызывает [OutOfRangeException](#), если исходный или целевой диапазоны находятся за пределами таблицы.

```
bool autoFill(const CellPosition& destination);
```

4.25.22.11 Метод `merge`

Служит для объединения ячеек в диапазоне.

```
void merge();
```

4.25.23 Класс `Cell`

Класс `Cell` представляет отдельную ячейку на листе электронной таблицы, либо ячейку таблицы в составе текстового документа.

```

class Cell
{
public:
    Range getRange();

    CellFormat getFormat() const;

    void setFormat(const CellFormat format);
    void setFormat(NumberCellFormatting format);
    void setFormat(PercentageCellFormatting format);
    void setFormat(CurrencyCellFormatting format);
    void setFormat(AccountingCellFormatting format);
    void setFormat(DateTimeCellFormatting format,
                CellFormat typeFormat = CellFormat::DateTime);
    void setFormat(FractionCellFormatting format);
    void setFormat(ScientificCellFormatting format);

    std::string getCustomFormat() const;
    void setCustomFormat(const std::string& formatString);

    void setBool(const bool value);
    void setNumber(const double value);
    void setText(const std::string& value);

    void setFormula(const std::string& value);
    std::string getFormulaAsString() const;
    std::string getFormattedValue() const;

    std::string getRawValue() const;

    void setFormattedValue(const std::string& value);

    void setContent(const std::string& value);

    CellProperties getCellProperties() const;
    void setCellProperties(const CellProperties& cellProperties);

    Borders getBorders();
    void setBorders(const Borders& borders);

    bool isPivotTableRoot() const;
    boost::optional<PivotTable> getPivotTable() const;

    ParagraphProperties getParagraphProperties() const;
    void setParagraphProperties(const ParagraphProperties& props);

    void unmerge();
};

```

4.25.23.1 Метод getRange

Возвращает диапазон [Range](#), позволяющий работать с содержимым ячейки как с абзацем ([Paragraph](#)) текста.

```
Range getRange();
```

4.25.23.2 Метод `getFormat`

Возвращает тип данных ячейки ([CellFormat](#): Общий, Числовой, Денежный, Текстовый и т.д.).

```
CellFormat getFormat() const;
```

4.25.23.3 Метод `setFormat`

Устанавливает тип данных ячейки ([CellFormat](#): Общий, Числовой, Денежный, Текстовый и т.д.).

```
void setFormat(const CellFormat format);  
void setFormat(NumberCellFormatting format);  
void setFormat(PercentageCellFormatting format);  
void setFormat(CurrencyCellFormatting format);  
void setFormat(AccountingCellFormatting format);  
void setFormat(DateTimeCellFormatting format, CellFormat typeFormat =  
CellFormat::DateTime);  
void setFormat(FractionCellFormatting format);  
void setFormat(ScientificCellFormatting format);
```

4.25.23.4 Метод `getCustomFormat`

Возвращает строку формата ячейки.

```
std::string getCustomFormat() const;
```

4.25.23.5 Метод `setCustomFormat`

Устанавливает формат ячейки. В случае невозможности установки формата ячейки вызывает исключение [ParseError](#).

```
void setCustomFormat(const std::string& formatString);
```

4.25.23.6 Методы `setBool`, `setNumber`, `setText`

Устанавливают для ячейки значение соответствующего типа.

```
void setBool(const bool value);  
void setNumber(const double value);  
void setText(const std::string& value);
```

4.25.23.7 Метод `setFormula`

Используется для ввода формулы в ячейку.

Формула – это любое выражение в ячейке, которое начинается со знака равенства (=). Формулы могут содержать функции, значения, адреса ячеек, имена, операторы действий и др.

Функция – это предустановленная формула приложения МойОфис Таблица, для вычисления которой необходимо использовать аргументы. Полный список функций

приведен в Приложении 1 «Перечень функций и их описание» в документе «МойОфис Стандартный. МойОфис Таблица. Руководство пользователя».

Основные принципы ввода формул и функций:

- формула всегда начинается со знака равенства (=);
- после знака равенства могут следовать функции, константы, адреса ячеек, операторы действий и другие элементы;
- все открывающие и закрывающие скобки должны быть согласованы;
- обязательные аргументы используемых функций должны быть указаны;
- константы не должны содержать символ «\$».

В случае невозможности ввода формулы вызывается исключение [ParseError](#).

```
void setFormula(const std::string& value);
```

4.25.23.8 Метод `getFormulaAsString`

Позволяет получить текст формулы в данной ячейке. Формула – это любое выражение в ячейке, которое начинается со знака равенства (=).

```
std::string getFormulaAsString() const;
```

4.25.23.9 Метод `getFormattedValue`

Возвращает значение ячейки, в виде строки, форматированной в соответствии с требованиями типа ячейки.

```
std::string getFormattedValue() const;
```

4.25.23.10 Метод `setFormattedValue`

Используется для установки значения ячейки, заранее форматированного в соответствии с требованиями типа ячейки. При невозможности сопоставить форматирование определенному типу данных используется форматирование для типа данных «Текстовый».

```
void setFormattedValue(const std::string& value);
```

4.25.23.11 Метод `getRowValue`

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

```
std::string getRowValue() const;
```


4.25.23.12 Метод `setContent`

Используется для установки значения ячейки и ее типа с использованием значения и типа параметра `value`. При невозможности сопоставить форматирование определенному типу данных используется форматирование для типа данных «Текстовый».

```
void setContent(const std::string& value);
```

4.25.23.13 Методы `getCellProperties`, `setCellProperties`

Методы для управления оформлением ячейки ([CellProperties](#): фон, вертикальное выравнивание и т. д.). Неустановленные свойства, которые имеют значение `boost::none` не будут изменены.

```
CellProperties getCellProperties() const;  
void setCellProperties(const CellProperties& cellProperties);
```

4.25.23.14 Метод `getBorders`

Возвращает границы отдельной ячейки.

```
Borders getBorders();
```

4.25.23.15 Метод `setBorders`

Устанавливает границы отдельной ячейки.

```
void setBorders(const Borders& borders);
```

4.25.23.16 Метод `isPivotTableRoot`

Возвращает `true`, если ячейка является корнем сводной таблицы.

```
bool isPivotTableRoot() const;
```

4.25.23.17 Метод `getPivotTable`

Возвращает сводную таблицу [PivotTable](#), если текущая ячейка входит в ее состав.

```
boost::optional<PivotTable> getPivotTable() const;
```

4.25.23.18 Методы `getParagraphProperties`, `setParagraphProperties`

Методы `getParagraphProperties` и `setParagraphProperties` предназначены для управления свойствами абзаца текста отдельной ячейки, такими как горизонтальное выравнивание текста, межстрочный интервал, межсимвольный интервал. В сочетании с методами `getCellProperties/setCellProperties` достигается возможность управления всеми настройками ячейки и ее содержимого.

```
ParagraphProperties getParagraphProperties() const;  
void setParagraphProperties(const ParagraphProperties& props);
```

4.25.23.19 Метод unmerge

Разъединяет несколько ячеек, которые были объединены ранее.

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

```
void unmerge();
```

4.25.24 Структура CellProperties

Структура CellProperties содержит настройки оформления ячейки.

```
struct CellProperties
{
    boost::optional<VerticalAlignment> verticalAlignment;
    boost::optional<ColorRGBA> backgroundColor;
    boost::optional<TextLayout> textLayout;
    boost::optional<TextOrientation> textOrientation;

    bool operator==(const& CellProperties& other) const;
    bool operator!=(const& CellProperties& other) const;
};
```

Структура содержит следующие поля:

- verticalAlignment – вертикальное выравнивание [VerticalAlignment](#);
- backgroundColor – цвет фона ячейки [ColorRGBA](#);
- textLayout – тип форматирования текста в ячейке [TextLayout](#);
- textOrientation – тип ориентации текста [TextOrientation](#).

4.25.24.1 Оператор ==

Оператор сравнения == предназначен для определения эквивалентности значений двух структур [CellProperties](#).

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

4.25.24.2 Оператор !=

Оператор сравнения != предназначен для определения неэквивалентности значений двух структур [CellProperties](#).

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

4.25.25 Структура `LineEndingProperties`

Структура `LineEndingProperties` содержит свойства оформления окончаний линий.

```
struct LineEndingProperties
{
    boost::optional<LineEndingStyle> style;
    boost::optional<Size<float>> relativeExtent;

    bool operator==(const& LineEndingProperties& other) const;
    bool operator!=(const& LineEndingProperties& other) const;
};
```

Структура содержит следующие поля:

- `style` – стиль окончания линии ([LineEndingStyle](#));
- `relativeExtent` – размер окончания линии относительно ее ширины.

4.25.25.1 Оператор `==`

Оператор сравнения `==` предназначен для определения эквивалентности значений двух структур [LineEndingProperties](#).

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

4.25.25.2 Оператор `!=`

Оператор сравнения `!=` предназначен для определения неэквивалентности значений двух структур [LineEndingProperties](#).

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

4.25.26 Структура `LineProperties`

Структура `LineProperties` содержит свойства оформления линий.

```
struct LineProperties
{
    boost::optional<LineStyle> style;
    boost::optional<LineWidth> width;
    boost::optional<Color> color;
    boost::optional<LineEndingProperties> headLineEndingProperties;
    boost::optional<LineEndingProperties> tailLineEndingProperties;

    bool operator==(const& LineProperties& other) const;
    bool operator!=(const& LineProperties& other) const;
};
```

Структура содержит следующие поля:

- `style` – стиль линии (см. [LineStyle](#));
- `width` – ширина линии;

- `color` – цвет линии (см. [Color](#));
- `headLineEndingProperties` – тип начала линии [LineEndingProperties](#);
- `tailLineEndingProperties` – тип конца линии [LineEndingProperties](#).

Удобная константа для невидимой линии (линия со стилем `NoLine`).

```
extern const LineProperties NoLineProperties;
```

4.25.26.1 Оператор `==`

Оператор сравнения `==` предназначен для определения эквивалентности значений двух структур `LineProperties`.

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

4.25.26.2 Оператор `!=`

Оператор сравнения `!=` предназначен для определения неэквивалентности значений двух структур `LineProperties`.

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

4.25.27 Структура `LineSpacing`

Структура `LineSpacing` позволяет управлять межстрочным интервалом.

```
struct LineSpacing
{
    LineSpacing(LineSpacingSize newSize, LineSpacingRule newRule);

    bool operator==(const LineSpacing& other) const;
    bool operator!=(const LineSpacing& other) const;

    LineSpacingSize value;
    LineSpacingRule rule;
};
```

4.25.27.1 Метод `LineSpacing`

Конструктор по умолчанию.

```
LineSpacing(LineSpacingSize newSize, LineSpacingRule newRule);
```

4.25.27.2 Оператор `==`

Оператор сравнения `==` используется для определения эквивалентности значений двух межстрочных интервалов.

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

4.25.27.3 Оператор `!=`

Оператор сравнения `!=` предназначен для определения неэквивалентности значений

двух межстрочных интервалов.

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

4.25.27.4 Поля структуры LineSpacing

Структура LineSpacing содержит следующие поля:

- value – значение межстрочного интервала;
- rule – тип межстрочного интервала (см. [LineSpacingRule](#)).

4.25.28 Класс Position

Класс Position обозначает местоположение в документе при вставке нового объекта.

```
class Position
{
public:
    void insertText(const std::string& text);

    Table insertTable(size_t rowCount,
                     size_t columnsCount,
                     const std::string& baseName);

    void insertPageBreak();
    void insertLineBreak();
    void insertBookmark(const std::string& name);
    void insertImage(const std::string& url,
                    const Size<size_t>& size);

    void insertSectionBreak();
    void insertSectionBreak(SectionBreakType breakType);

    void removeBackward(size_t count = 1u);
    void removeForward(size_t count = 1u);

    bool operator==(const Position& other) const;
    bool operator!=(const Position& other) const;
};
```

4.25.28.1 Метод insertText

Вставляет текст в указанную позицию.

```
void insertText(const std::string& text);
```

4.25.28.2 Метод insertTable

Вставляет таблицу в указанную позицию. При невозможности вставки таблицы вызывает исключение [DocumentModificationError](#).

```
Table insertTable(size_t rowCount, size_t columnsCount,
                  const std::string& baseName);
```

4.25.28.3 Метод `insertPageBreak`

Вставляет разделитель страниц в указанную позицию.

```
void insertPageBreak();
```

4.25.28.4 Метод `insertLineBreak`

Вставляет переход на следующую строку, не разрывающий абзац, в указанную позицию.

```
void insertLineBreak();
```

4.25.28.5 Метод `insertBookmark`

Вставляет закладку с наименованием `name` в текущую позицию.

```
void insertBookmark(const std::string& name);
```

4.25.28.6 Метод `insertImage`

Вставляет рисунок, размещенный в файле, в текущую позицию. С помощью параметра `url` задается полный путь к файлу. Параметр `size` задает геометрические размеры изображения для вставки.

```
void insertImage(const std::string& url, const Size<float>& size);
```

4.25.28.7 Метод `operator==`

Оператор сравнения `==` предназначен для определения эквивалентности значений двух местоположений в документе.

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

4.25.28.8 Метод `operator!=`

Оператор сравнения `!=` предназначен для определения неэквивалентности значений двух местоположений в документе.

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

4.25.28.9 Метод `insertSectionBreak`

Вставляет в позицию разрыв раздела.

```
void insertSectionBreak();  
void insertSectionBreak(SectionBreakType breakType);
```

4.25.28.10 Метод `removeBackward`

Удаляет `count` объектов (символов, картинок и т.д.) до текущей позиции.

```
void removeBackward(size_t count = 1u);
```

4.25.28.11 Метод `removeForward`

Удаляет `count` объектов (символов, картинок и т.д.) после текущей позиции.

```
void removeForward(size_t count = 1u);
```

4.25.29 Класс `Range`

Класс `Range` управляет информацией о диапазоне позиций в документе и предоставляет начальную и конечную позицию.

```
class Range
{
public:
    Range(const Position& begin, const Position& end);

    Position getBegin() const;
    Position getEnd() const;

    std::string extractText() const;

    void removeContent();

    void lockContent();
    void unlockContent();
    bool isContentLocked();

    void replaceText(const std::string& text);

    TextProperties getTextProperties() const;

    void setTextProperties(const TextProperties& textProperties);

    std::shared_ptr<Enumerator<Block>> getBlocksEnumerator();
    std::shared_ptr<Enumerator<TrackedChange>> getTrackedChangesEnumerator();

    Comments getComments();
    Paragraphs getParagraphs();
    Images getImages();
    MediaObjects getInlineObjects();
};
```

4.25.29.1 Метод `Range`

Конструктор – создает объект `Range` для указанного в параметрах диапазона.

```
Range(const Position& begin, const Position& end);
```

4.25.29.2 Метод `getBegin`

Возвращает начальную позицию диапазона, значение только для чтения.

```
Position getBegin() const;
```

4.25.29.3 Метод `getEnd`

Возвращает конечную позицию диапазона, значение только для чтения.

```
Position getEnd() const;
```

4.25.29.4 Метод `extractText`

Возвращает текстовое представление содержимого диапазона. При этом часть содержимого (например, фотографии) будет пропущена, часть (например, таблицы) будет преобразована.

```
std::string extractText() const;
```

4.25.29.5 Метод `removeContent`

Удаляет содержимое диапазона.

```
void removeContent();
```

4.25.29.6 Метод `lockContent`

Метод запрещает изменения содержимого диапазона. В случае попытки изменения вызывается исключение [DocumentModificationError](#).

```
void lockContent();
```

4.25.29.7 Метод `unlockContent`

Метод разрешает изменения содержимого диапазона.

```
void unlockContent();
```

4.25.29.8 Метод `isContentLocked`

Метод возвращает значение `true`, если изменения содержимого диапазона запрещены.

```
bool isContentLocked();
```

4.25.29.9 Метод `replaceText`

Заменяет содержимое диапазона на указанный текст.

```
void replaceText(const std::string& text);
```

4.25.29.10 Метод `getTextProperties`

Возвращает свойства [TextProperties](#) оформления текста в диапазоне. Свойства, которые имеют смешанные значения, будут неустановленными (`boost::none`).

```
TextProperties getTextProperties() const;
```


4.25.29.11 Метод `setTextProperties`

Настройка оформления текста [TextProperties](#). Неустановленные свойства (которые имеют значение `boost::none`) не будут изменены.

```
void setTextProperties(const TextProperties& textProperties);
```

4.25.29.12 Метод `getBlocksEnumerator`

Возвращает объект типа `Enumerator` для доступа к коллекции блоков в диапазоне.

```
std::shared_ptr<Enumerator<Block>> getBlocksEnumerator();
```

4.25.29.13 Метод `getTrackedChangesEnumerator`

Возвращает объект типа `Enumerator` для доступа к коллекции отслеживаемых изменений.

```
std::shared_ptr<Enumerator<TrackedChange>> getTrackedChangesEnumerator();
```

4.25.29.14 Метод `getComments`

Обеспечивает доступ к комментариям [Comments](#) в диапазоне. Комментарии, примененные к одному и тому же диапазону, упорядочиваются по датам, если таковые имеются. Если дат нет, то порядок комментариев не определен.

```
Comments getComments();
```

4.25.29.15 Метод `getParagraphs`

Обеспечивает доступ к абзацам [Paragraphs](#) в диапазоне.

```
Paragraphs getParagraphs();
```

4.25.29.16 Метод `getImages`

Обеспечивает доступ к изображениям [Images](#) в диапазоне.

```
Images getImages();
```

4.25.29.17 Метод `getInlineObjects`

Обеспечивает доступ к встроенным фигурам в диапазоне.

```
InlineObjects getInlineObjects();
```

4.25.30 Класс `Search`

Класс `Search` предоставляет интерфейс для выполнения поиска в документе.

```
class Search
{
public:
    virtual std::shared_ptr<Enumerator<Range>> findText(const std::string& text) = 0;
};
```

```
virtual std::shared_ptr<Enumerator<Range>> findText(const std::string& text,
                                                    const Range& range) = 0;
};
```

4.25.30.1 Метод findText

Обеспечивает поиск текста в документе без учета регистра.

```
virtual std::shared_ptr<Enumerator<Range>> findText(const std::string& text) =
0;
```

Обеспечивает поиск текста в указанном диапазоне, без учета регистра.

```
virtual std::shared_ptr<Enumerator<Range>> findText(const std::string& text,
                                                    const Range& range) = 0;
```

4.25.30.2 Глобальная функция createSearch

Создает новый объект [Search](#) в указанном объекте [Document](#).

```
std::shared_ptr<Search> createSearch(const Document& document);
```

4.25.31 Структура TextProperties

Структура `TextProperties` содержит свойства оформления фрагмента текста (диапазона). Это может быть либо весь абзац, либо его часть, либо даже текст в нескольких ячейках в таблице. Свойства оформления текста могут быть применены к любому объекту [Range](#).

```
struct TextProperties
{
    boost::optional<std::string> fontName;
    boost::optional<FontSize> fontSize;
    boost::optional<bool> bold;
    boost::optional<bool> italic;
    boost::optional<bool> underline;
    boost::optional<bool> strikethrough;
    boost::optional<bool> allCapitals;
    boost::optional<ScriptPosition> scriptPosition;
    boost::optional<Color> textColor;
    boost::optional<ColorRGBA> backgroundColor;
    boost::optional<CharacterSpacingSize> characterSpacing;
};
```

Структура содержит следующие поля:

- `fontName` – наименование шрифта;
- `fontSize` – размер шрифта;
- `bold` – принимает значение `true`, если шрифт текста «полужирный»;
- `italic` – принимает значение `true`, если шрифт текста «курсив»;
- `underline` – принимает значение `true`, если шрифт текста «подчеркнутый»;
- `strikethrough` – принимает значение `true`, если шрифт текста «зачеркнутый»;

- `allCapitals` – принимает значение `true`, если в тексте все символы прописные;
- `scriptPosition` – тип надстрочного/подстрочного форматирования;
- `textColor` – цветовая модель оформления текста [Color](#);
- `backgroundColor` – цветовая модель фона текста [ColorRGBA](#);
- `characterSpacing` – межзнаковый интервал.

4.25.32 Класс `Bookmarks`

Класс `Bookmarks` предоставляет интерфейс для доступа к закладкам документа.

```
class Bookmarks
{
public:
    boost::optional<Range> getBookmarkRange(const std::string& bookmarkName);
    void removeBookmark(const std::string& bookmarkName);
};
```

4.25.32.1 Метод `getBookmarkRange`

Возвращает диапазон, содержащий закладку с наименованием `bookmarkName` или `boost::none`, если нет такой закладки.

```
boost::optional<Range> getBookmarkRange(const std::string& bookmarkName);
```

4.25.32.2 Метод `removeBookmark`

Удаляет закладку с именем `bookmarkName`.

```
void removeBookmark(const std::string& bookmarkName);
```

4.25.33 Класс `Comment`

Класс `Comment` предоставляет доступ к свойствам комментария, примененного к диапазону.

```
class Comment
{
public:
    Range getRange() const;

    boost::optional<bool> isResolved() const;

    boost::optional<std::string> getText() const;

    boost::optional<std::string> getAudioUrl() const;

    Comments getReplies() const;

    TrackedChangeInfo getInfo() const;
};
```

4.25.33.1 Метод `getRange`

Возвращает объект [Range](#), соответствующий объекту [Comment](#).

```
Range getRange() const;
```

4.25.33.2 Метод `isResolved`

Возвращает `true`, если комментарий решен.

```
boost::optional<bool> isResolved() const;
```

4.25.33.3 Метод `getText`

Возвращает текст комментария.

```
boost::optional<std::string> getText() const;
```

4.25.33.4 Метод `getAudioUrl`

Возвращает путь к файлу аудиокomentarия.

```
boost::optional<std::string> getAudioUrl() const;
```

4.25.33.5 Метод `getReplies`

Предоставляет доступ к ответам на комментарии ([Comments](#)).

```
Comments getReplies() const;
```

4.25.33.6 Метод `getInfo`

Предоставляет доступ к отслеживаемой информации об изменении комментария ([TrackedChangeInfo](#): автор изменения, дата и т. д.).

```
TrackedChangeInfo getInfo() const;
```

4.25.34 Класс `Comments`

Класс `Comments` предоставляет интерфейс для доступа к коллекции комментариев диапазона.

```
class Comments
{
public:
    std::shared_ptr<Enumerator<Comment>> getEnumerator();
};
```

4.25.34.1 Метод `getEnumerator`

Возвращает объект типа `Enumerator` для доступа к коллекции комментариев.

```
std::shared_ptr<Enumerator<Comment>> getEnumerator();
```

4.25.35 Класс TrackedChange

Класс TrackedChange предоставляет интерфейс для отслеживания изменений в документе.

```
class TrackedChange
{
public:
    TrackedChange(std::shared_ptr<TrackedChangeImpl>&& impl);

    Range getRange() const;

    TrackedChangeType getType() const;
    const TrackedChangeInfo& getInfo() const;
};
```

4.25.35.1 Метод TrackedChange

Конструктор по умолчанию.

```
TrackedChange(std::shared_ptr<TrackedChangeImpl>&& impl);
```

4.25.35.2 Метод getRange

Возвращает объект [Range](#), который соответствует измененному диапазону внутри абзаца.

```
Range getRange() const;
```

4.25.35.3 Методы getType

Позволяет получить тип отслеживаемого изменения [TrackedChangeType](#).

```
TrackedChangeType getType() const;
```

4.25.35.4 Методы getInfo

Позволяет получить информацию об отслеживаемых изменениях [TrackedChangeInfo](#).

```
const TrackedChangeInfo& getInfo() const;
```

4.25.36 Структура TrackedChangeInfo

Структура TrackedChangeInfo содержит информацию об отслеживаемых изменениях.

```
struct TrackedChangeInfo
{
    boost::optional<UserInfo> author;
    boost::optional<DateTime> timeStamp;

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

```
bool operator!=(const TrackedChangeInfo& other) const;
};
```

4.25.36.1 Поля структуры TrackedChangeInfo

Структура TrackedChangeInfo содержит следующие поля:

- author – автор изменений;
- timeStamp – дата и время изменений [DateTime](#).

4.25.36.2 Оператор ==

Оператор сравнения == используется для определения эквивалентности двух отслеживаемых изменений.

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

4.25.36.3 Оператор !=

Оператор сравнения != используется для определения неэквивалентности двух отслеживаемых изменений.

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

4.25.37 Класс Section

Класс Section предоставляет методы для управления отдельными разделами (sections) в документе.

```
class Section
{
    void setPageProperties(const PageProperties& properties);
    PageProperties getPageProperties() const;
    void setPageOrientation(PageOrientation orientation);
    boost::optional<PageOrientation> getPageOrientation() const;
    HeadersFooters getHeaders();
    HeadersFooters getFooters();
    Range getRange();
};
```

4.25.37.1 Метод setPageProperties

Устанавливает высоту и ширину [PageProperties](#) страниц раздела.

```
void setPageProperties(const PageProperties& properties);
```

4.25.37.2 Метод getPageProperties

Возвращает размеры высоты и ширины [PageProperties](#) для страниц раздела.

```
PageProperties getPageProperties() const;
```

4.25.37.3 Метод `setPageOrientation`

Задаёт ориентацию [PageOrientation](#) страниц раздела.

```
void setPageOrientation(PageOrientation orientation);
```

4.25.37.4 Метод `getPageOrientation`

Возвращает ориентацию страниц раздела [PageOrientation](#) или значение `boost::none`, если ориентация не установлена.

```
boost::optional<PageOrientation> getPageOrientation() const;
```

4.25.37.5 Метод `getRange`

Возвращает диапазон [Range](#) в документе, соответствующий данному разделу.

```
Range getRange();
```

4.25.37.6 Метод `getHeaders`

Предоставляет доступ к коллекции верхних колонтитулов [HeadersFooters](#), содержащихся в разделе.

```
HeadersFooters getHeaders();
```

4.25.37.7 Метод `getFooters`

Предоставляет доступ к коллекции нижних колонтитулов [HeadersFooters](#), содержащихся в разделе.

```
HeadersFooters getFooters();
```

4.25.38 Класс `Sections`

Класс для представления списка разделов документа.

```
class Sections
{
public:
    std::shared_ptr<Enumerator<Section>> getEnumerator();
};
```

4.25.38.1 Метод `getEnumerator`

Возвращает объект `Enumerator` для доступа к коллекции разделов документа.

```
std::shared_ptr<Enumerator<Section>> getEnumerator();
```

4.25.39 Класс `SectionBreakType`

Тип `SectionBreakType` определяет тип начала следующей секции после вставки разрыва раздела.

```
enum class SectionBreakType : std::uint8_t
{
```

```
NextPage,  
Continious,  
EvenPage,  
OddPage  
};
```

Поддерживаются следующие типы разрывов разделов документа:

- NextPage – следующий раздел начинается с новой страницы;
- Continious – следующий раздел продолжается на текущей странице без вставки разрыва страницы;
- EvenPage – следующий раздел начинается на ближайшей четной странице;
- OddPage – следующий раздел начинается на ближайшей нечетной странице.

4.25.40 Класс PageOrientation

Тип PageOrientation определяет варианты ориентации страницы документа: Альбомная (Landscape) или Книжная (Portrait).

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

4.25.41 Структура PageProperties

Структура PageProperties содержит размеры (height, width) и поля (margins) страницы.

```
struct PageProperties  
{  
    boost::optional<Insets> margins;  
    boost::optional<Unit> height = .0f;  
    boost::optional<Unit> width = .0f;  
  
    bool operator==(const PageProperties& other);  
    bool operator!=(const PageProperties& other);  
};
```

4.25.41.1 Оператор ==

Оператор сравнения == используется для определения эквивалентности двух структур PageProperties.

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

4.25.41.2 Оператор !=

Оператор сравнения != используется для определения неэквивалентности двух структур PageProperties.


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

4.25.42 Структура Insets

Структура Insets содержит поля страницы (left, right, top, bottom).

```
struct Insets
{
    boost::optional<Unit> left;
    boost::optional<Unit> right;
    boost::optional<Unit> top;
    boost::optional<Unit> bottom;

    bool operator==(const Insets& other) const;
    bool operator!=(const Insets& other) const;
};
```

4.25.42.1 Оператор ==

Оператор сравнения == используется для определения эквивалентности двух структур Insets.

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

4.25.42.2 Оператор !=

Оператор сравнения != используется для определения неэквивалентности двух структур Insets.

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

4.25.43 Класс HeaderComponent

Класс HeaderComponent определяет колонтитул документа.

```
class HeaderComponent
{
public:
    HeaderComponentType getType() const;
    Blocks getBlocks();
    Range getRange();
};
```

4.25.43.1 Метод getType

Предоставляет информацию о типе колонтитула [HeaderFooterType](#).

```
HeaderFooterType getType() const;
```

4.25.43.2 Метод getBlocks

Предоставляет доступ к блокам, которые содержатся в колонтитуле.

```
Blocks getBlocks();
```

4.25.43.3 Метод `getRange`

Предоставляет диапазон с содержанием верхнего или нижнего колонтитулов.

```
Range getRange();
```

4.25.44 Класс `HeadersFooters`

Класс `HeadersFooters` представляет коллекцию верхних и нижних колонтитулов в разделе (`Section`) документа. См. описание класса [Section](#).

```
class HeadersFooters
{
public:
    std::shared_ptr<Enumerator<HeaderFooter>> getEnumerator();
};
```

4.25.44.1 Метод `getEnumerator`

Возвращает объект `Enumerator` для доступа к коллекции колонтитулов.

```
std::shared_ptr<Enumerator<HeaderFooter>> getEnumerator();
```

4.25.45 Класс `TextOrientation`

Класс `TextOrientation` управляет свойством ориентации текста в ячейке, фигуре и т.д.

```
class TextOrientation
{
public:
    TextOrientation() = default;
    explicit TextOrientation(double degrees);

    boost::optional<double> getAngle() const;
    bool isStackedChars() const;

    bool operator==(const TextOrientation& other) const;
    bool operator!=(const TextOrientation& other) const;
};
```

4.25.45.1 Метод `TextOrientation`

Конструктор по умолчанию.

```
TextOrientation() = default;
```

Конструктор с ориентацией текста в ячейке под заданным углом. Значение угла указывается в градусах.

```
explicit TextOrientation(double degrees);
```

4.25.45.2 Метод `getAngle`

Возвращает угол ориентации текста в ячейке или `boost::none`, если он не установлен.

```
boost::optional<double> getAngle() const;
```

4.25.45.3 Метод `isStackedChars`

Возвращает `true`, если ориентация текста - вертикальный столбец.

```
bool isStackedChars() const;
```

4.25.45.4 Оператор `==`

Оператор сравнения `==` предназначен для определения эквивалентности значений двух структур `TextOrientation`.

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

4.25.45.5 Оператор `!=`

Оператор сравнения `!=` предназначен для определения неэквивалентности значений двух структур `TextOrientation`.

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

4.25.46 Структура `TextExportSettings`

Структура `TextExportSettings` содержит настройки для экспорта текстовых документов. См. [PageNumbers](#).

```
struct TextExportSettings
{
    PageNumbers pageNumbers = PageNumbers{};
};
```

4.25.47 Структура `WorkbookExportSettings`

Структура `WorkbookExportSettings` содержит настройки для экспорта табличных документов.

```
struct WorkbookExportSettings
{
    SheetNames sheetNames = SheetNames{};
    PrintingScope printingScope = PrintingScope{};
    PageProperties pageProperties = PageProperties{};
    Percents scale = 100.0f;
};
```

Структура `WorkbookExportSettings` содержит следующие поля:

- `sheetNames` – представляет коллекцию имен листов для экспорта [SheetNames](#). Если коллекция пуста, экспортируются все листы;
- `printingScope` – представляет область печати [PrintingScope](#) (весь документ, область печати, пользовательский диапазон и т. д.);
- `pageProperties` – представляет свойства страницы для экспортируемого документа [PageProperties](#) (высота/ширина страницы и т.д.);
- `scale` – представляет масштаб экспорта документа в процентах (например, 50,0%, 150,63%, 400,0% и т. д.) (см. [Percents](#)).

4.25.48 Класс `PrintingScope`

Класс `PrintingScope` содержит настройки для экспорта табличных документов.

```
class PrintingScope
{
public:
    enum class Type : uint8_t
    {
        PrintArea,
        WholeSheet
    };

    PrintingScope();
    explicit PrintingScope(Type type);
    explicit PrintingScope(const CellRangePosition& range);

    bool usePrintArea() const;

    boost::optional<CellRangePosition> getCellRange() const;
};
```

4.25.48.1 Класс `PrintingScope::Type`

Класс `PrintingScope::Type` представляет собой predefined тип области печати.

```
enum class Type : uint8_t
{
    PrintArea,
    WholeSheet
};
```

Класс `PrintingScope::Type` содержит следующие поля:

- `PrintArea` – представляет выбранную область печати;
- `WholeSheet` – представляет область печати – весь документ.

4.25.48.2 Метод `PrintingScope`

Конструктор по умолчанию. Создает области печати типа `PrintArea`.

```
PrintingScope();
```

Конструктор для создания области печати типа `PrintArea` или `WholeSheet`.

```
explicit PrintingScope(Type type);
```

Конструктор для создания области печати типа [CellRangePosition](#).

```
explicit PrintingScope(const CellRangePosition& range);
```

4.25.48.3 Метод `usePrintArea`

Метод возвращает `true`, если область печати должна использоваться во время печати, экспорта и т. д.

```
bool usePrintArea() const;
```

4.25.48.4 Метод `getCellRange`

Метод возвращает диапазон [CellRangePosition](#) ячеек таблицы или `boost::none`.

```
boost::optional<CellRangePosition> getCellRange() const;
```

4.25.49 Класс `PageNumbers`

Класс `PageNumbers` представляют собой коллекцию страниц для экспорта (нечетные, четные, список и т. д.).

```
class PageNumbers
{
public:
    PageNumbers();
    explicit PageNumbers(PageParity parity);
    explicit PageNumbers(std::vector<size_t> pageNumbers);
    PageNumbers(size_t firstPageNumber, size_t lastPageNumber);

    bool contains(size_t pageNumber) const;
    boost::optional<size_t> getLast() const;
};
```

4.25.49.1 Метод `PageNumbers`

Конструктор по умолчанию для представления всех страниц.

```
PageNumbers();
```

Конструктор для четных – нечетных страниц (см. [PageParity](#)).

```
explicit PageNumbers(PageParity parity);
```

Конструктор для списка номеров страниц.

```
explicit PageNumbers(std::vector<size_t> pageNumbers);
```

Конструктор для диапазона номеров страниц.

```
PageNumbers(size_t firstPageNumber, size_t lastPageNumber);
```

4.25.49.2 Метод contains

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

```
bool contains(size_t pageNumber) const;
```

4.25.49.3 Метод getLast

Метод getLast возвращает последний номер страницы.

```
boost::optional<size_t> getLast() const;
```

4.25.50 Класс Color

Класс Color представляет либо цветовой объект RGBA, либо заданные цвета идентификатора темы.

```
class Color
{
    Color();
    Color(const ColorRGBA& rgbaColor);
    Color(const ThemeColorID themeColorID);

    const boost::optional<ColorRGBA> getRGBAColor() const;
    const boost::optional<ThemeColorID> getThemeColorID() const;

    void setTransforms(const ColorTransforms& transforms);
    const boost::optional<ColorTransforms> getTransforms() const;

    bool operator==(const Color& other) const;
    bool operator!=(const Color& other) const;
};
```

4.25.50.1 Метод Color

Конструктор по умолчанию.

```
Color();
```

Конструктор создает [ColorRGBA](#) объект.

```
Color(const ColorRGBA& rgbaColor);
```

Конструктор создает объект с цветами заданного идентификатора темы (см. [ThemeColorID](#)).

```
Color(const ThemeColorID themeColorID);
```

4.25.50.2 Метод `getRGBAColor`

Метод возвращает цвет [ColorRGBA](#) или `boost::none`.

```
const boost::optional<ColorRGBA> getRGBAColor() const;
```

4.25.50.3 Метод `getThemeColorID`

Метод возвращает цвет идентификатора темы [ThemeColorID](#) или `boost::none`.

```
const boost::optional<ThemeColorID> getThemeColorID() const;
```

4.25.50.4 Метод `setTransforms`

Метод устанавливает правило трансформации цвета, реализованное интерфейсом [ColorTransforms](#).

```
void setTransforms(const ColorTransforms& transforms);
```

4.25.50.5 Метод `getTransforms`

Метод возвращает объект [ColorTransforms](#), реализующий трансформацию цвета.

```
const boost::optional<ColorTransforms> getTransforms() const;
```

4.25.50.6 Оператор `==`

Оператор сравнения `==` предназначен для определения эквивалентности значений двух цветовых моделей, включая значение прозрачности.

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

4.25.50.7 Оператор `!=`

Оператор сравнения `!=` предназначен для определения неэквивалентности значений двух цветовых моделей, включая значение прозрачности.

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

4.25.51 Класс `ColorTransforms`

Интерфейс `ColorTransforms` предназначен для конвертации цвета и содержит метод `apply`, конвертирующий цвет.

```
class ColorTransforms
{
public:
    ColorRGBA apply(const ColorRGBA& color) const;
};
```

4.25.52 Класс Frame

Класс `Frame` представляет прямоугольную область встроенного объекта, расположенного как символ в тексте. Класс предназначен для изменения положения и геометрии встроенного объекта, управления переносом текста вокруг объекта.

```
class Frame
{
    void setPosition(const TextAnchoredPosition& position);
    boost::optional<AnchoredPosition> getPosition() const;

    void setDimensions(const boost::optional<Size<float>>& size);
    boost::optional<Size<float>> getDimensions() const;

    void setWrapType(TextWrapType type);
    boost::optional<TextWrapType> getWrapType() const;
};
```

4.25.52.1 Метод `setPosition`

Метод задает положение встроенного объекта. Для текстовых документов позиция может быть установлена только для встроенных объектов, тип переноса текста, которых не является `TextWrapType::Inline`.

```
void setPosition(const TextAnchoredPosition& position);
```

4.25.52.2 Метод `getPosition`

Метод возвращает позицию встроенного объекта на странице, если позиция не определена, то возвращает `boost::none`.

```
boost::optional<AnchoredPosition> getPosition() const;
```

4.25.52.3 Метод `getDimensions`

Возвращает размеры встроенного объекта или `boost::none`, если размеры не определены.

```
boost::optional<Size<Unit>> getDimensions() const;
```

4.25.52.4 Метод `setDimensions`

Метод задает размеры (изменяет размер) встроенного объекта. Размеры считаются неопределенными, если их значение равно `boost::none`.

```
void setDimensions(const boost::optional<Size<Unit>>& size);
```

4.25.52.5 Метод `setWrapType`

Устанавливает вариант обтекания текстом встроенного объекта [TextWrapType](#).

```
void setWrapType(TextWrapType type);
```


4.25.52.6 Метод `getWrapType`

Возвращает вариант обтекания текстом встроенного объекта [TextWrapType](#).

```
boost::optional<TextWrapType> getWrapType() const;
```

4.25.53 Класс `Image`

Класс `Image` представляет собой изображение, как встроенный объект.

```
class Image
{
    Frame getFrame();
};
```

4.25.53.1 Метод `getFrame`

Метод возвращает прямоугольную область встроенного объекта [Frame](#).

```
Frame getFrame();
```

4.25.54 Класс `Images`

Класс `Images` предоставляет интерфейс для доступа к коллекции изображений.

```
class Images
{
    std::shared_ptr<Enumerator<Image>> getEnumerator();
};
```

4.25.54.1 Метод `getEnumerator`

Возвращает объект `Enumerator` для доступа к коллекции изображений.

```
std::shared_ptr<Enumerator<Image>> getEnumerator();
```

4.25.55 Структура `RelativeAnchoredPosition`

Структура `RelativeAnchoredPosition` предоставляет собой шаблон класса для управления относительным положением объекта со смещением или выравниванием.

```
struct RelativeAnchoredPosition
{
    using RelativeToType = typename TraitsType::RelativeToType;
    using AlignmentType = typename TraitsType::AlignmentType;

    RelativeAnchoredPosition(RelativeToType relativeTo,
                             Unit offset = .Of);
    RelativeAnchoredPosition(RelativeToType relativeTo,
                             AlignmentType alignment);

    bool operator==(const RelativeAnchoredPosition& other) const;
    bool operator!=(const RelativeAnchoredPosition& other) const;

    RelativeToType relativeTo;
    boost::optional<Unit> offset;
```

```
boost::optional<AlignmentType> alignment;  
};
```

Структура `RelativeAnchoredPosition` содержит следующие поля:

– `relativeTo` – значение [HorizontalRelativeTo](#) или [VerticalRelativeTo](#);

– `offset` – значение смещения;

– `alignment` – значение [HorizontalAnchorAlignment](#) или [VerticalAnchorAlignment](#).

4.25.55.1 Метод `RelativeAnchoredPosition`

Конструктор создает объект с заданным типом относительного позиционирования `relativeTo` и значением смещения `offset`.

```
RelativeAnchoredPosition(RelativeToType relativeTo, Unit offset = .Of);
```

Конструктор создает объект с заданным типом относительного позиционирования `relativeTo` и значением выравнивания `alignment`.

```
RelativeAnchoredPosition(RelativeToType relativeTo, AlignmentType alignment);
```

4.25.55.2 Оператор `==`

Оператор сравнения `==` используется для определения эквивалентности значений двух относительных позиций.

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

4.25.55.3 Оператор `!=`

Оператор сравнения `!=` используется для определения неэквивалентности значений двух относительных позиций.

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

4.25.56 Структура `TextAnchoredPosition`

Структура `TextAnchoredPosition` представляет позицию объекта на странице текстового документа.

```
struct TextAnchoredPosition  
{  
    TextAnchoredPosition();  
    TextAnchoredPosition(Unit horizontalOffset, Unit verticalOffset);  
    TextAnchoredPosition(const HorizontalTextAnchoredPosition& horizontal,  
                          const VerticalTextAnchoredPosition& vertical);  
  
    bool operator==(const TextAnchoredPosition& other) const;  
    bool operator!=(const TextAnchoredPosition& other) const;  
  
    HorizontalTextAnchoredPosition horizontal;
```

```
VerticalTextAnchoredPosition vertical;  
};
```

Структура содержит следующие поля:

- `horizontal` – позиция по горизонтали [HorizontalTextAnchoredPosition](#);
- `vertical` – позиция по вертикали [VerticalTextAnchoredPosition](#).

4.25.56.1 Метод `TextAnchoredPosition`

Конструктор по умолчанию создает объект с координатами (0, 0) по горизонтали относительно столбца и по вертикали относительно верхней границы страницы.

```
TextAnchoredPosition();
```

Конструктор создает объект с позицией с заданными смещениями `horizontalOffset` по горизонтали относительно столбца и `verticalOffset` по вертикали относительно верхней границы страницы.

```
TextAnchoredPosition(Unit horizontalOffset, Unit verticalOffset);
```

Конструктор создает объект с заданными координатами `horizontal` по горизонтали и `vertical` по вертикали.

```
TextAnchoredPosition(const HorizontalTextAnchoredPosition& horizontal,  
                    const VerticalTextAnchoredPosition& vertical);
```

4.25.56.2 Оператор `==`

Оператор сравнения `==` используется для определения эквивалентности значений двух позиций.

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

4.25.56.3 Оператор `!=`

Оператор сравнения `!=` используется для определения неэквивалентности значений двух позиций.

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

4.25.57 Структура `AnchoredPosition`

Структура `AnchoredPosition` представляет позицию с относительными смещениями на странице текстового документа.

```
struct AnchoredPosition  
{  
    AnchoredPosition(const TextAnchoredPosition& textPosition);  
  
    bool operator==(const AnchoredPosition& other) const;
```

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

boost::optional<TextAnchoredPosition> textPosition;
};
```

Структура содержит следующее поле:

- textPosition – позиция на странице текстового документа [TextAnchoredPosition](#).

4.25.57.1 Метод AnchoredPosition

Конструктор по умолчанию создает объект с закрепленной позицией в текстовом документе.

```
AnchoredPosition(const TextAnchoredPosition& textPosition);
```

4.25.57.2 Оператор ==

Оператор сравнения == используется для определения эквивалентности значений двух позиций.

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

4.25.57.3 Оператор !=

Оператор сравнения != используется для определения неэквивалентности значений двух позиций.

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

4.25.58 Класс InlineObject

Класс InlineObject представляет собой встроенный объект, который позиционируется как символ в строке текста.

```
class InlineObject
{
public:
    boost::optional<Image> toImage();

    Frame getFrame();
};
```

4.25.58.1 Метод toImage

Метод возвращает изображение [Image](#), связанное со встроенным объектом. В случае ошибки возвращает boost::none.

```
boost::optional<Image> toImage();
```

4.25.58.2 Метод `getFrame`

Метод возвращает прямоугольную область встроеного объекта [Frame](#).

```
Frame getFrame();
```

4.25.59 Класс `InlineObjects`

Класс `InlineObjects` предоставляет интерфейс для доступа к коллекции встроенных объектов.

```
class InlineObjects
{
public:
    std::shared_ptr<Enumerator<InlineObject>> getEnumerator();
};
```

4.25.59.1 Метод `getEnumerator`

Возвращает объект `Enumerator` для доступа к коллекции встроенных объектов.

```
std::shared_ptr<Enumerator<InlineObject>> getEnumerator();
```

4.26 Классы и методы для работы с макрокомандами (`Scripting`)

4.26.1 Класс `Scripting`

Класс `Scripting` управляет виртуальной машиной Lua. Он предоставляет интерфейс для выполнения макрокоманд Lua, которые хранятся в электронном документе.

```
class Scripting
{
public:
    virtual std::string runScript(const std::string& name) = 0;
};
```

4.26.1.1 Метод `runScript`

Запускает макрокоманду с указанным именем. В случае невозможности запуска макрокоманды вызывает исключение [ScriptExecutionError](#).

```
virtual void runScript(const std::string& name) = 0;
```

4.26.1.2 Глобальная функция `createScripting`

Создает объект `Scripting` для доступа к макрокомандам в электронном документе.

```
std::shared_ptr<Scripting> createScripting(const Document::Document& document);
```

4.27 Исключения

4.27.1 Класс BaseError

Класс BaseError является базовым классом для всех исключений SDK.

```
class BaseError : public std::runtime_error
{
public:
    explicit BaseError(const std::string& error);
};
```

4.27.2 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);
};
```

4.27.3 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);
};
```

4.27.4 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);
};
```

4.27.5 Класс DocumentCreateError

Исключение DocumentCreateError вызывается в случае, когда документ не может быть создан.

```
class DocumentCreateError : public BaseError
{
public:
```

```
explicit DocumentCreateError(const std::string& error);  
};
```

4.27.6 Класс DocumentLoadError

Исключение DocumentLoadError вызывается в случае, когда документ не может быть загружен.

```
class DocumentLoadError : public BaseError  
{  
public:  
    explicit DocumentLoadError(const std::string& error);  
};
```

4.27.7 Класс DocumentSaveError

Исключение DocumentSaveError вызывается в случае, когда документ не может быть сохранен.

```
class DocumentSaveError : public BaseError  
{  
public:  
    explicit DocumentSaveError(const std::string& error);  
};
```

4.27.8 Класс DocumentExportError

Исключение DocumentExportError вызывается в случае, когда документ не может быть экспортирован.

```
class DocumentExportError : public BaseError  
{  
public:  
    explicit DocumentExportError(const std::string& error);  
};
```

4.27.9 Класс NoSuchElementError

Исключение NoSuchElementError вызывается в случае, когда элемент не существует.

```
class NoSuchElementError : public BaseError  
{  
public:  
    NoSuchElementError();  
};
```

4.27.10 Класс NotImplementedError

Исключение NotImplementedError вызывается в случае, если обнаружена нереализованная функциональность.

```
class NotImplementedError : public BaseError  
{
```

```
public:
    explicit NotImplementedError(const std::string& error);
};
```

4.27.11 Класс `OutOfRangeException`

Исключение `OutOfRangeException` вызывается в случае обнаружения выхода значения за пределы диапазона.

```
class OutOfRangeError : public BaseError
{
public:
    explicit OutOfRangeError(const std::string& rangeAsStr);
};
```

4.27.12 Класс `ParseError`

Исключение `ParseError` вызывается в случае, когда текст не прошел синтаксический анализ.

```
class ParseError : public BaseError
{
public:
    explicit ParseError(const std::string& error);
};
```

4.27.13 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);
};
```

4.27.14 Класс `ForbiddenActionError`

Исключение `ForbiddenActionError` вызывается в случае выполнения запрещенной операции.

```
class ForbiddenActionError : public BaseError
{
public:
    explicit ForbiddenActionError(const std::string& reason);
};
```


4.27.15 Класс DocumentModificationError

Исключение DocumentModificationError вызывается в случае, когда невозможно выполнить операцию по изменению документа.

```
class DocumentModificationError : public BaseError
{
    public:
        DocumentModificationError(const std::string& reason);
};
```

4.27.16 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);
};
```

4.27.17 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();
};
```

4.27.18 Класс ScriptExecutionError

Исключение ScriptExecutionError вызывается в случае, когда сценарий не удастся выполнить.

```
class ScriptExecutionError : public BaseError
{
    public:
        explicit ScriptExecutionError(const std::string& error);
};
```

5 ВЕРСИИ DOCUMENT API

5.1 Механизм контроля версий

Константы версии Document API Major и Minor позволяют проверить совместимость предыдущей и текущей версии Document API.

Если была изменена константа Major версии Document API, то в Document API произошли обратно несовместимые изменения, и программный код должен быть пересмотрен и обновлен. Обратно несовместимыми изменениями считаются: переименование, удаление или несовместимое изменение подписи существующих классов или методов, а также добавление новых методов, типов и полей класса.

В разделе [Изменения](#) указывается добавлены ли несовместимые изменения в Document API.

Если была изменена константа Minor версии Document API, то в Document API произошли только обратно совместимые изменения, и нет необходимости менять программный код, чтобы он работал с более новой версией Document API.

Но гарантируется совместимость только на уровне исходного кода C++, поэтому необходимо перекомпилировать программный код приложения с более новой версией Document API.

Рекомендуется проверить версию Document API до инициализации, как указано ниже:

```
unsigned int ExpectedMajorAPIVersion = 1;
unsigned int ExpectedMinorAPIVersion = 0;
if (!Version::isAPIVersionCompatible(ExpectedMajorAPIVersion,
ExpectedMinorAPIVersion))
{
// Вывод сообщения о серьезной ошибке несовместимости версии библиотеки Document
API и выход из программы
}
// Работа с библиотекой Document API (создание объекта Application и т. д.)

Текущие версии Document API:
const unsigned int Major = 1;
const unsigned int Minor = 0;
```

Пример проверки совместимости указанной версии Document API с текущей:

```
bool isAPIVersionCompatible(unsigned int major, unsigned int minor);
```

Указанные в примере константы и функции определены в пространстве имени Version.

5.2 Изменения

Обратно несовместимые изменения в Document API отсутствуют.