Методы Crafttech Power Extension

- createStyledTable
- createXlsx
- decryptFile
- deleteActiveFileDrawings
- deleteDrawings
- deleteFile
- deleteRowFile
- deleteSheet
- deleteTempDirForce
- dropRowFile
- editCellRange
- editCells
- editCellsValue
- editCellValue
- editSheetPrintOptions
- editSheetTabColor
- editSheetVisibility
- editTxtFile
- encryptFile
- executeMacros
- existDir
- existFile
- findCellsBySubString
- findCellsByValue
- freezeField
- getActiveSheetName
- getAllCharts
- getAllChartsBySheet
- getBaseDir
- getCellRange
- getCellValue
- getColumnIndex

- getCurrentFilePath
- getDefaultCheckboxValues
- getDefaultRadioboxValues
- getDirPathsByDialog
- getDocTables
- getFieldLength
- getFileData
- getFilesInDir
- getHtml
- getListBox
- getPageCount
- getPathsByDialog
- getSheetArray
- getSheetsNames
- getStyledTable
- getUsedRange
- getUserName
- goalSeek
- groupActive
- hideColumns
- hideRows
- infoFile
- inputForm
- insertRowFile
- isFileEncrypted
- isFileOpen
- isMergedCell
- isSheetExist
- isSheetProtected
- macroStatus
- mathFunction
- mathFunctionMany

- mathRand
- mergeCells
- moveSheetName
- numberFormat
- openExplorer
- openFile
- pasteDataFromBuffer
- protectBookActive
- protectFile
- protectRangeActive
- protectSheet
- protectSheetActive
- recalcAllFormulas
- removeAutoFilter
- renameFile
- renameSheet
- reRenderSheet
- resetAutofilterActive
- resetFilters
- resizeActiveFileStyledTable
- runFile
- saveActiveFile
- saveFile
- selectRangeDialog
- sendByMail
- sendMail
- setAutofilterActive
- setAutoFitCell
- setCellSize
- showColumns
- showInputBox
- showLevels

- showRows
- sortActiveSheet
- spinner
- unfreezeField
- ungroupActive
- unmergeCells
- unProtectFile
- unprotectRangeActive
- unProtectSheet
- updateActiveFilePivotTable
- updateStyledTable
- XML
    - Методы
    - Методы

Математические-методы

Здесь представлен список математических функций и аргументов для них:

```
"beta_inv": (probability: number, alpha: number, beta_param: number,
min_param=None, max_param=None)
```

- Возвращает значение переменной x, для которого функция распределения бета-распределения равна заданной вероятности probability. Диапазон можно задать через min_param и max_param.

```
"beta_dist": (kpi: number, alpha: number, beta_param: number, accumulated:
bool, min_param: number, max_param: number)
```

- Возвращает значение функции распределения бета-распределения для заданного значения kpi, параметров формы alpha, beta_param и диапазона. Если accumulated = true, считает накопленную вероятность (CDF).

```
"norm_s_inv": (p: number)
```

- Возвращает обратное стандартное нормальное значение (среднее = 0, σ = 1) для заданной вероятности p. Это квантиль стандартного нормального распределения.

```
"log_norm_dist": (x: number, mean: number, std_dev: number, cumulative: bool)
```

- Возвращает значение функции (PDF или CDF) для логнормального распределения с заданными mean и std_dev при значении x. Если cumulative = true — возвращает накопленную вероятность (CDF), иначе плотность (PDF).

```
"norm_s_dist": (z: number, cumulative: bool)
```

- Возвращает значение функции (PDF или CDF) для стандартного нормального распределения при заданном z. Если cumulative = true — возвращает CDF, иначе — PDF.

```
"log_norm_inv": (p: number, mean: number, std_dev: number)
```

- Возвращает значение переменной x для которого CDF логнормального распределения с параметрами mean и std_dev равна заданной вероятности p

---

```
"norm_dist": (x: number, mean: number, std_dev: number, cumulative: bool)
```

- Возвращает значение функции (PDF или CDF) для нормального распределения с заданными mean, std_dev при значении x. Если cumulative = true — возвращает CDF, иначе — PDF.

```
"norm_inv": (probability: number, mean: number, std_dev: number)
```

- Возвращает значение переменной x для которого CDF нормального распределения с параметрами mean и std_dev равна заданной вероятности probability.

addMacros

```
await craftTechApi.addMacros(fileHandler, macrosPath, macrosName);
```

Описание

Добавляет новый макрос в список макросов в Р7-Офис.

Параметры

- **fileHandler** (number) — ссылка на файл, куда нужно добавить макрос; ссылку предоставляет метод openFile;
- **macrosPath** (string) — путь до файла с макросом с расширением `.js`;
- **macrosName** (string) — название передаваемого макроса.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного добавления нового макроса.

Пример

```
const result = await craftTechApi.addMacros(fileHandler, 'macro.js', 'New
macro');
console.log('Макрос добавлен: ', result);
```

applyActiveFileChanges

```
await craftTechApi.applyActiveFileChanges(fileHandler, iconType = 'alert',
title = 'Внимание!',
    message = "Для завершения работы макроса вам необходимо закрыть документ в Р7-
офис и других программах.
    После сохранения файла он будет заново открыт.", width = 'auto');
```

Описание

Применяет изменения, внесенные методами Crafttech PE в активный файл и хранящиеся в базе данных папки temp.

> Нажатие на “ОК” самостоятельно закроет файл и откроет заново

> Данный метод не сохранит изменения, внесенные нативным API Р7-Офис. Для корректной работы необходимо сначала внести все изменения при помощи API Р7-Офис, затем вызвать `saveActiveFile()`, после чего открыть файл при помощи `openFile()` и использовать методы Crafttech PE.

> Данный метод необходимо вызывать в конце макроса, так как он прекращает его выполнение.

> Файл, который сохраняется при помощи `applyActiveFileChanges()` НЕ нужно закрывать при помощи `closeFile()`.

Возвращает

Метод ничего не возвращает, поскольку закрывает файл с макросом.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.
- **iconType** (string) — не обязательный параметр, указывающий тип уведомления.
- **title** (string) — не обязательный параметр, указывающий заголовок уведомления.
- **message** (string) — не обязательный параметр, указывающий сообщение уведомления.

- **width** (string) — не обязательный параметр, указывающий ширину уведомления.

Пример функции без указания доп. параметров

```
await craftTechApi.applyActiveFileChanges(fileHandler);
```

Пример функции с указанием параметров

```
await craftTechApi.applyActiveFileChanges(handler, 'alert', 'Внимание!', 'Файл закроется!!!','auto' )
```

autoRecalcFormula

```
await craftTechApi.autoRecalcFormula(fileHandler);
```

Описание

Запускает автопересчёт формул при каждом открытии документа.

Параметры

- **fileHandler** (number) — ссылка на файл.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного пересчёта формул в файле, иначе `false`.

Пример

```
const result = await craftTechApi.autoRecalcFormula(fileHandler);
console.log('Выполнен автопересчёт: ', result);
```

clearCellRange

```
await craftTechApi.clearCellRange(
  handler,
  sheet_name,
  cell_range,
  flag='value, formula, merge, style'
);
```

Описание

Очищает данные в диапазоне.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **handler** (number) — ссылка на файл.
- **sheet_name** (string) — название листа.
- **cell_range** (string) — диапазон ячеек.
- **flag** (string) — данные, которые мы хотим зачистить, по умолчанию - ‘value, formula, merge, style’.

Возвращает

- **true | false** (boolean)

Пример

```
const fh = await craftTechApi.openFile('book.xlsx');
const isClear = await craftTechApi.clearCellRange(fh, 'Лист1', 'A1:B2')
console.log('Диапазон очищен: ', isClear);
```

clearCurrentCells

```
craftTechApi.clearCurrentCells(sheet, range, flag = 'all')
```

Описание

Очищает значение ячейки в активном файле, из которого запускается макрос.

> Он был разработан для обхода ошибки Р7-Офис, при котором SetValue(”) не очищает ячейку, а присваивает ей значение ‘0’.

Параметры

- **sheet** (ApiWorksheet) — объект, в котором проводится действие;
- **range** (string) — диапазон ячеек, в котором проводится действие;
- **flag** (string) — флаг, который указывает, какую именно информацию необходимо очистить:
  - all (по умолчанию) — всё содержимое ячейки;
  - text — только текст;
  - format — только форматирование;
  - comments — только комментарий;
  - hyperlinks — только гиперссылки.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.clearCurrentCells(Api.GetSheet('Лист1'), 'A1:A3', 'text');
```

Код выше очистит текст в ячейках A1, A2 и A3, сохранив при этом их форматирование.

closeFile

```
await craftTechApi.closeFile(fileHandler);
```

Описание

Закрывает внешний файл после окончания работы с ним.

> Крайне важно **всегда** закрывать все внешние файлы по завершению работы макроса (если в ходе выполнения они открывались через openFile), иначе макрос может вести себя непредсказуемо, а любые изменения внешних файлов не будут сохраняться.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного закрытия файла, иначе `false`.

Пример

```
const fh = await craftTechApi.openFile('book.xlsx');
console.log('ID файла: ', fh);
await craftTechApi.closeFile(fh);
```

conditionalFormatting

```
await craftTechApi.conditionalFormatting(fileHandler, sheetName, config);
```

Описание

Добавляет условное форматирование для выбранных ячеек.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для текущего файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **config** (object) — объект с полями, которые необходимо поменять. Поля могу сильно розниться в зависимости от типа. Смотрите необходимый вам тип для заполнения конфига. Основные поля:
  - sqref (обязательный) (string) — ячейка или диапазон ячеек, к которому будет применено условное форматирование;
  - type (обязательный) (string) — тип условного форматирования, см. Типы условного форматирования ниже;
  - style (object) — объект стилей, для заполнения см. editCells;

Типы условного форматирования

containsText

срабатывает, если ячейка включает в себя текст. Для корректной работы необходимо добавить в `config` поле `text`.

Пример конфига:

```
{
   sqref: 'A1',
   text: '+',
   style: {
      font: {
         colorRgb: 'FFDADADA',
         patternType: 'solid',
      }
   },
   type: 'containsText',
}
```

cells

для работы необходимо указать в `config` поля `operator`, а также `formula`. В формулу передается число для сравнения со значением ячейки. Доступны следующие операторы:

1. `greaterThan` (больше чем),
2. `greaterThanOrEqual` (больше либо равно),
3. `lessThan` (меньше чем),
4. `lessThanOrEqual` (меньше либо равно),
5. `equal` (равно),
6. `notEqual` (не равно),
7. `between` (между, в поле `formula` необходимо передать массив с двумя числами),
8. `notBetween` (не между, в поле `formula` необходимо передать массив с двумя числами).

Пример конфига:

```
{
   sqref: 'A1',
   style: {
      font: {
         colorRgb: 'FFDADADA',
         patternType: 'solid',
      }
   },
   type: 'cellIs',
   operator: 'notEqual',
   formula: 0,
}
```

iconSet

изменяется в зависимости от значения ячейки. Для корректной работы необходимо добавить в `config` поля: `values`, `iconSet`, `showValue`(необязательный).

Пример конфига:

> `iconSet`: 3Symbols (крест, воскл. знак, галочка в круге), 3Symbols2 (крест, воскл. знак, галочка без круга)

> укажите `showValue: false`, если хотите, чтобы отображался только значок (без значения ячейки) порядок

> элементов values (условий) важен при выборе значка, т.е. на примере 3Symbols2: первое относится к кресту, второе - к воскл. знаку, третье - к галочке.

> указать `gte: false` в элементе массива `values` необходимо, если вы хотите использовать строгое сравнение (< вместо <=)

```
let config = {
  sqref: 'A1',
  type: 'iconSet',
  showValue: false,
  iconSet: '3Symbols2',
  values: [
    {
      type: 'percent', // num, percent (численное, либо процентное значение)
      val: 0,
    },
    {
      type: 'num',
      gte: false,
      val: 2,
    },
    {
      type: 'num',
      gte: false,
      val: 3,
    },
  ]
}
```

expression

работает на версии **1.0.2.80.2 и выше.**

срабатывает, если удовлетворяет значению формулы. Для корректной работы необходимо добавить в `config` поле `formula` (может быть массивом формул).

Пример конфига:

```
{
    style: {
        font: {
            colorRgb: 'FF9C5700'
        },
        fill: {
            fgRgb: 'FFFFEB9C',
            bgRgb: 'FFFFEB9C',
            patternType: 'solid',
        }
    },
    sqref: 'A1',
    type: 'expression',
    formula: 'E5 > 0'
}
```

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае успешного применения условного форматирования, иначе `false`.

Пример

```
const conf = {
   sqref: 'A1',
   text: '+',
   style: {
      font: {
         colorRgb: 'FFDADADA',
         patternType: 'solid',
      }
   },
   type: 'containsText',
}

const result = await craftTechApi.conditionalFormatting(fileHandler, 'Лист1',
conf);
console.log('Результат выполнения: ', result);
```

convertCellsToBoxes

```
await craftTechApi.convertCellsToBoxes({checkboxes:[{range: string, flag?:
boolean}], radioboxes:[{range: string, checked?: string}]});
```

Описание

Формирует чекбоксы и радиобоксы на место указанных ячеек в активном файле, из которого запущен текущий макрос.

> Данный метод был разработал ввиду ограничений Р7-Офис Таблицы, не предусматривающих создание и взаимодействие с пользовательскими формами и элементами управления. Единственный выход, который мы можем предложить заказчикам — явно имитировать действие чекбоксов и радиобоксов через диапазон ячеек.

> Макрос, использующий данный метод, должен работать в формате автозапуска (А).

> Можно использовать вместо предыдущих версий методов конвертации, просто указывая какой-то один массив.

> Для получения символов (да, нет) использовать методы: -getDefaultCheckboxValues - getDefaultRadioboxValues

```
await craftTechApi.convertCellsToBoxes({checkboxes:[{range: "Лист1!A1:A5",
flag: false}]})
```

Параметры

- **checkboxes** (Array) — массив объектов, на основе которых формируются чекбоксы; у каждого объекта есть свойства:
  - **range** (string) одного диапазона, с указанием листа, в строковом формате ("Лист1!A1:F1");
  - **flag?** (boolean) необязательный параметр; флаг, указывающий, должны ли активизироваться все чекбоксы в диапазоне при нажатии на одну из них (то есть «выбрать всё»); если не указано, то по умолчанию ставится false

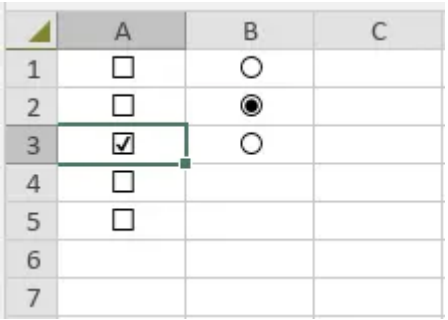
- **radioboxes** (Array) — массив объектов, на основе которых формируются радиобоксы; у каждого объекта есть свойства:
  - **range** (string) одного диапазона, с указанием листа, в строковом формате (`"Лист1!A1:F1"`);
  - **checked?** (string) необязательный параметр; адрес ячейки, указывающий, где автоматически должно быть отмечено (“C1”); если не указано, возьмётся первая ячейка в диапазоне (“A1”).

Возвращает

Метод ничего не возвращает.

Пример

```
await craftTechApi.convertCellsToBoxes({checkboxes:[{range: "Лист1!A1:A5",
flag: false}], radioboxes:[{range: "Лист1!B1:B3", checked: "B2"}]});
```



convertCellToCheckbox

```
await craftTechApi.convertCellToCheckbox(range, options, flag);
```

Описание

Формирует чекбоксы (флажки) на место указанных ячеек в активном файле, из которого запущен текущий макрос.

> Данный метод был разработал ввиду ограничений Р7-Офис Таблицы, не предусматривающих создание и взаимодействие с пользовательскими формами и элементами управления.

> Не путать с convertCellToRadiobox, который подразумевает **единичный** выбор опций. В convertCellToCheckbox можно выбрать несколько ячеек из диапазона.

> Макрос, использующий данный метод, должен работать в формате автозапуска (A).

Параметры

- **range** (string[] | string) — диапазон ячеек, на основе которых формируются чекбоксы; принимаются диапазоны в виде:
  - одного диапазона в строковом формате (`"A1:F1"`);
  - массива строк с адресами ячеек (`["A1", "B1", "C1", "D1", "E1", "F1"]`);
  - микс адресов ячеек и диапазонов ячеек в виде массива (`["A1", "B1:B4", "C1", "D1", "E1", "F1:F2", "A5:C5"]`);
- **options** (string[]) — необязательный параметр; массив с двумя строками, означающие бинарные опции с состояниями `true` или `false`, соответственно (то есть, если чекбокс активен, то ячейка принимает значение первого элемента параметра, если неактивен, то второго):
  - если параметр не передаётся или равен пустой строке, на месте ячеек формируются классические чекбоксы в привычном виде: квадрат, внутри которого отмечена или не отмечена галочка в зависимости от состояния;
- **flag** (string) — необязательный параметр; флаг, указывающий, должны ли активизироваться все чекбоксы в диапазоне при нажатии на одну из них (то есть

«выбрать всё»):

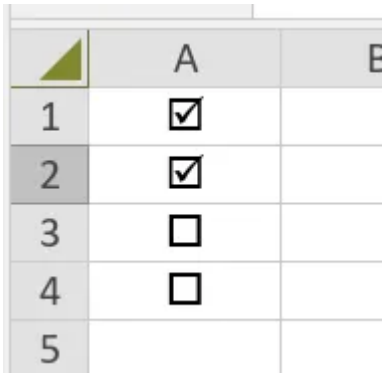
- all — выбрать всё.

Возвращает

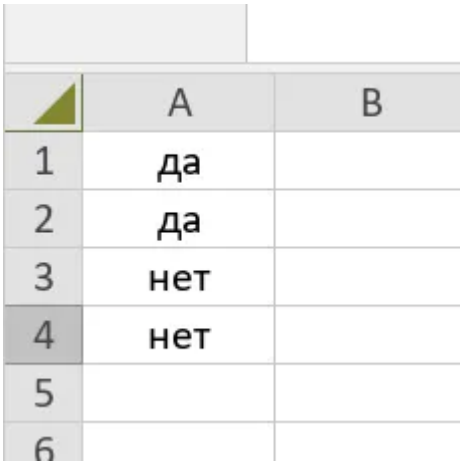
Метод ничего не возвращает.

Пример

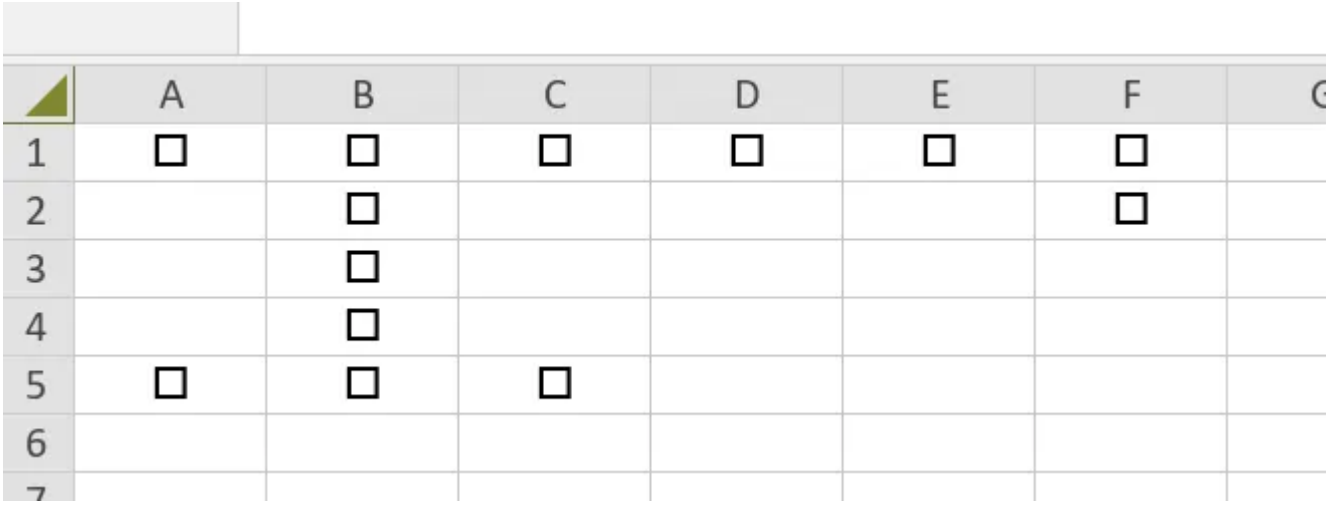
```
await craftTechApi.convertCellToCheckbox('A1:A4');
```




```
await craftTechApi.convertCellToCheckbox('A1:A4', ['да', 'нет']);
```




```
await craftTechApi.convertCellToCheckbox(["A1", "B1:B4", "C1", "D1", "E1",
"F1:F2", "A5:C5"]);
```



convertCellToRadiobox

```
await craftTechApi.convertCellToRadiobox(range, options);
```

Описание

Формирует радиобоксы (радиокнопки) на место указанных ячеек в активном файле, из которого запущен текущий макрос.

> Данный метод был разработал ввиду ограничений Р7-Офис Таблицы, не предусматривающих создание и взаимодействие с пользовательскими формами и элементами управления.

> Не путать с convertCellToCheckbox, который подразумевает **множественный** выбор опций. В convertCellToRadiobox можно выбрать только одну ячейку из диапазона.

> Макрос, использующий данный метод, должен работать в формате автозапуска (A).

Параметры

- **range** (string[] | string) — диапазон ячеек, на основе которых формируются радиобоксы; принимаются диапазоны в виде:
  - одного диапазона в строковом формате (`"A1:F1"`);
  - массива строк с адресами ячеек (`["A1", "B1", "C1", "D1", "E1", "F1"]`);
  - микс адресов ячеек и диапазонов ячеек в виде массива (`["A1", "B1:B4", "C1", "D1", "E1", "F1:F2", "A5:C5"]`);
  - диапазоны ячеек с указанием листов (если название листа содержит проблемы, то его нужно указывать в виде одинарных кавычек):`["Лист1!A1", "'Расчет года'!B1:B4"]`
- **options** (string[] | object[]) — необязательный параметр; массив, содержащий бинарные опции с состояниями `true` или `false`, соответственно (то есть, если радиобокс активен, то ячейка принимает значение первого элемента параметра, если неактивен, то второго); три случая:
  - string[] — передаём массив строк, если бинарные опции представляют собой строковые значения, например:

        - `['да', 'нет']`,
        - `['yes', 'no']`,
        - `['ok', 'fail']`,
        - `['v', 'x']`,
        - и так далее;
    - object[] — передаём массив объектов с конфигурацией стилей если бинарные опции представляют собой форматирование ячеек: изменение цвета, установка границ и так далее;
        - для установки стилей необходимо придерживаться конкретной структуры: для каждого стиля задаётся поле, соответствующее методу Р7-Офис, а в качестве значения — параметры метода в виде массива массивов;
    - если параметр не передаётся или равен пустой строке, на месте ячеек формируются классические чекбоксы в привычном виде: квадрат, внутри которого отмечена или не отмечена галочка в зависимости от состояния.

Если вам необходимо, чтобы на листе было несколько радиобоксов и они работали независимо друг от друга, передавайте адреса ячеек массивом. Если нужно, чтобы на листе работал только один радиобокс вне зависимости от их количества, передавайте диапазон строкой, либо массивом с одним элементом.

Пример бинарных опций, когда нам нужно передать стили:

```
// ячейка ДА
const yesCell = {
  SetBorders:
    [
      ["DiagonalDown", "Thin", Api.CreateColorFromRGB(0, 0, 0)],
      ["DiagonalUp", "Thin", Api.CreateColorFromRGB(0, 0, 0)]
    ],
  SetFillColor:
    [
      [Api.CreateColorFromRGB(255, 213, 191)]
    ],
};

// ячейка НЕТ
const noCell = {
  SetFillColor: [
    [Api.CreateColorFromRGB(255, 213, 191)]
  ],
};
```

Возвращает

Метод ничего не возвращает.

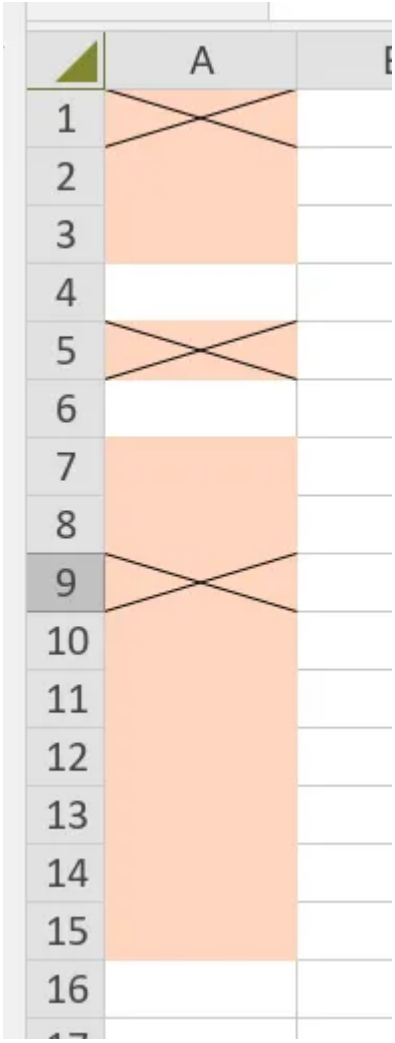
Пример

```
// ячейка ДА
const yesCell = {
  SetBorders:
    [
      ["DiagonalDown", "Thin", Api.CreateColorFromRGB(0, 0, 0)],
      ["DiagonalUp", "Thin", Api.CreateColorFromRGB(0, 0, 0)]
    ],
  SetFillColor:
    [
      [Api.CreateColorFromRGB(255, 213, 191)]
    ],
};

// ячейка НЕТ
const noCell = {
  SetFillColor: [
    [Api.CreateColorFromRGB(255, 213, 191)]
  ],
};

const choice = [yesCell, noCell];

await craftTechApi.convertCellToRadiobox(['A1:A3', 'A5', 'A7:A15'], choice);
```



Код выше устанавливает на диапазонах `'A1:A3'`, `'A5'` и `'A7:A15'` радиобоксы. Неактивные ячейки содержат заливку персикового цвета, активные — дополнительно границы по диагонали. Активные ячейки — `A1`, `A5`, `A9`; неактивные — все остальные.

При этом каждый из диапазонов, переданных массивом, работают независимо друг от друга. Если нажать на ячейку `A12`, то ячейка `A9` перестанет быть активной, поскольку они принадлежат одному диапазону, в то время как ячейки `A1` и `A5` останутся в прежнем состоянии, так как за ним закреплены другие диапазона.

convertDate

```
await craftTechApi.convertDate(badDate);
```

Описание

Преобразовывает дату из формата Р7-Офис в читаемый формат.

> Он был разработан для обхода ошибки Р7-Офис, при котором даты определяется в виде отсчёта от 01.01.1900 (например, Р7-Офис преобразовывает дату `27.04.2009` в `39930`). Такой результат возвращается как при использовании методов API Р7-Офис, так и методов микросервиса. Цель этого метода — преобразовать это число в читаемый формат.

> Также в числе может содержаться время, например, `39930.5` - это `27.04.2009 12:00:00`, то есть середина дня.

Параметры

- **badDate** (number) — дата в формате Р7-Офис.

Возвращает

- **goodDate** (object) — объект с датой в читаемом формате:
  - date (string | false) — дата в строковом формате, либо `false`;
  - time (string | false) — время в строковом формате, либо `false`.

```
{
  date: '27.04.2009',
  time: '12:00:00'
}
```

Пример

```
const myDate = await craftTechApi.convertDate(39930.5);
console.log('Дата: ', myDate.date, ' время: ', myDate.time);
```

convertPdfToBase64

```
await craftTechApi.convertPdfToBase64(pdfPath);
```

Описание

Конвертирует PDF-файл в строку формата Base64.

> Подробнее про Base64 можно почитать тут.

Параметры

- **pdfPath** (string) — путь до pdf-файла.

Возвращает

- **base64path** (string) — строку, закодированную в формат Base64.

Пример

```
const result = await craftTechApi.convertPdfToBase64('files/test.pdf');
console.log('convertPdfToBase64(): ', result);
```

convertTime

```
await craftTechApi.convertTime(badTime);
```

Описание

Преобразовывает время из формата Р7-Офис в читаемый формат.

> Он был разработан для обхода бага Р7-Офис, при котором время определяется в виде отсчёта от 0 (например, Р7-Офис преобразовывает дату `21:59:59` в `0.916666`). Такой результат возвращается как при использовании методов API Р7-Офис, так и методов микросервиса. Цель этого метода — преобразовать это число в читаемый формат.

Параметры

- **`badTime`** (number) — время в формате Р7-Офис.

Возвращает

- **goodTime** (object) — объект со временем в читаемом формате:
  - time (string) — время в строковом формате.

```
{
  time: '21:59:59'
}
```

Пример

```
const myTime = await craftTechApi.convertTime(0.916666);
console.log('Время: ', myTime.time);
```

convertToPdf

```
await craftTechApi.convertToPdf(filePath, sheetName, pdfPath);
```

Описание

Конвертирует содержимое листа xlsx-файла в PDF-файл.

Параметры

- **filePath** (string) — путь до файла со старым именем;
- **sheetName** (string) — название листа;
- **pdfPath** (string) — путь до pdf-файла, который мы создаём.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешной конвертации листа в PDF-файл, иначе `false`.

Пример

```
const result = await craftTechApi.convertToPdf('Test.xlsx', 'Лист1',
'files/test.pdf');
console.log('convertToPdf(): ', result);
```

copyCellRange

```
await craftTechApi.copyCellRange(
  fileHandlerSource,
  sheetNameSource,
  rangeSource,
  fileHandlerTarget,
  sheetNameTarget,
  rangeTarget,
  flag='value, style, formula
')
```

Описание

Копирует диапазон листа книги.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandleSource** (number) — ссылка на файл, из которого мы копируем лист; ссылку предоставляет метод openFile;
- **sheetNameSource** (string) — название листа, из которого копируется диапазон;
- **rangeSource** (string) — копируемый диапазон; адреса ячеек разделяются двоеточием (например: `A1:C2`, либо `1,1:2,3`; во втором случае сначала передаётся строка, а потом столбец);
- **fileHandlerTarget** (number) — ссылка на файл, в который копируется диапазон; если мы хотим скопировать диапазон в этот же файл, fileHandlerTarget должен совпадать с fileHandlerSource;
- **sheetNameTarget** (string) — название листа, в который копируется диапазон;
- **rangeTarget** (string) — диапазон, в который копируются ячейки; адреса ячеек разделяются двоеточием (например: `A1:C2`, либо `1,1:2,3`; во втором случае сначала передаётся строка, а потом столбец);

- **flag** (string|{ insert: string, clear: string}) — необязательный параметр, указывающий, какие данные ячейки нужно скопировать:
  - `value` — только значения;
  - `style` — только стили.
  - `formula` — только формулы.
  - флаги можно комбинировать, перечисляя их через запятую, например: `value, formula`.
  - также можно указать какие данные мы хотим удалить в редактируемом диапазоне:

```
{
  "insert": "value, style, formula",
  "clear": "value, formula, merge, style"
}
```

Возвращает

- **true | false** (boolean) — возвращает логическую истину в случае успешного копирования диапазона.

Пример

```
const check = await craftTechApi.copyCellRange(
  fhSource,
  'Лист1',
  'A1:A3',
  fhTarget,
  'Лист1',
  'C1:C3', //   '1,3:3,3',
  'value, style'
)
```

После выполнения данного кода из исходного файла скопируются ячейки `A1:A3` и вставятся в другой файл в диапазон `C1:C3` с сохранением стилей.

Пример переноса из 1й ячейки

```
const check = await craftTechApi.copyCellRange(
  fhSource,
  'Лист1',
  'A1',
  fhTarget,
  'Лист1',
  'C1:C3', //   '1,3:3,3',
  'value, style, formula'
)
```

После выполнения данного кода из исходного файла скопируются ячейки `A1` и вставятся в другой файл в диапазон `C1:C3` с сохранением стилей.

copyCurrentWorksheet

```
craftTechApi.copyCurrentWorksheet(sheetStr, newSheetStr);
```

Описание

Копирует лист в активном файле, из которого запущен текущий макрос.

Параметры

- **sheetStr** (string) — название листа, который мы копируем;
- **newSheetStr** (string) — название нового листа, который мы скопировали.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.copyCurrentWorksheet('Лист1', 'Лист1Копия');
```

copyDir

```
await craftTechApi.copyDir(sourceDirPath, targetDirPath);
```

Описание

Копирует папку со всеми файлами внутри в указанное место.

Параметры

- **sourceDirPath** (string) — путь до копируемой папки;
- **targetDirPath** (string) — путь до папки, в которую копируем (или путь до новой папки, которой не существует).

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного копирования папки, иначе `false`.

Пример

```
const isCopied = await craftTechApi.copyDir('test', 'hello/test');
console.log('Папка скопирована: ', isCopied);
```

copyFile

```
await craftTechApi.copyFile(sourceFilePath, targetFilePath);
```

Описание

Копирует файл в указанное место.

Параметры

- **sourceFilePath** (string) — путь до копируемого файла;
- **targetFilePath** (string) — путь до копии файла (или путь до нового файла, которого не существует).

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного копирования файла, иначе `false`.

Пример

```
const isCopied = await craftTechApi.copyFile('test.txt', 'testCopy.txt');
console.log('Файл скопирован: ', isCopied);
```

copySheet

```
await craftTechApi.copySheet(fileHandlerOld, sheetNameOld, fileHandlerNew,
sheetNameNew, flag='value, style');
```

Описание

Копирует лист книги.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для текущего файла).

Параметры

- **fileHandlerOld** (number) — ссылка на файл, из которого мы копируем лист; ссылку предоставляет метод openFile;
- **sheetNameOld** (string) — название листа, который мы копируем;
- **fileHandlerNew** (number) — ссылка на файл, в который мы копируем лист; если мы хотим скопировать лист в этот же файл, fileHandlerNew должен совпадать с fileHandlerOld;
- **sheetNameNew** (string) — название нового листа, который мы скопировали.
- **flag** (string) — (по умолчанию ‘value, style’), указывает какие данные мы хотим взять с листа, можно указать value, style и formula.

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае успешного копирования листа, иначе `false`.

Пример

```
const check = await craftTechApi.copySheet(fileHandler1, 'Лист1', fileHandler2,
'Лист1Копия');
console.log('copySheet(): ', check);
```

copySizeCell

```
craftTechApi.copySizeCell(cell_1, cell_2, sheetStr)
```

Описание

Копирует размер (ширину и высоту) одной ячейки и вставляет его в другую в активном файле, из которого запущен текущий макрос.

Параметры

- **cell_1** (string) — адрес ячейки, из которой будем копировать ширину и высоту;
- **cell_2** (string) — адрес ячейки, в которую будем копировать ширину и высоту;
- **sheetStr** (string) — название листа, в котором проводится действие.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.copySizeCell('A3', 'A4', 'Лист1');
```

Пусть ячейка A3 имеет ширину 100 у.е. и длину 20 у.е., а A4 имеет размер относительно длины текста. После выполнения кода выше ячейка A4 приобретёт те же свойства, что и A3: ширину 100 у.е. и длину 20 у.е.

copyStyleCell

```
craftTechApi.copyStyleCell(cell_1, cell_2, sheetStr)
```

Описание

Копирует стили одной ячейки и вставляет их в другую в активном файле, из которого запущен текущий макрос.

Параметры

- **cell_1** (string) — адрес ячейки, из которой будем копировать стили;
- **cell_2** (string) — адрес ячейки, в которую будем копировать стили.
- **sheetStr** (string) — название листа, в котором проводится действие.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.copyStyleCell('A1', 'A2', 'Лист1');
```

createAutoFilter

```
craftTechApi.createAutoFilter(handler, sheetName, range)
```

Описание

Устанавливает фильтрацию по диапазону значений.

> Для активного файла можно использовать setAutofilterActive;

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа на котором нужно установить автофильтр;
- **range** (string) — диапазон ячеек, которому необходимо установить автофильтр.

Возвращает

true | false (boolean) — логическая истина `true` в случае успешного удаления листа, иначе `false`.

Пример

```
craftTechApi.createAutoFilter(handler, 'Лист1', 'A1:D15');
```

createDir

```
await craftTechApi.createDir(newDirPath);
```

Описание

Создаёт папку по указанному пути и добавляет её в систему.

Параметры

- **newDirPath** (string) — путь до новой папки.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного создания папки, иначе `false`.

Пример

```
const isCreated = await craftTechApi.createDir('hello/there/myDir');
console.log('Папка создана: ', isCreated);
```

createDropdownCell

```
await craftTechApi.createDropdownCell(fileHandler, sheetName, cellAddress,
dropdownRange);
```

Описание

Преобразовывает ячейки в выпадающий список, использующий данные по ссылке.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для текущего файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellAddress** (string) — адрес ячейки (или диапазона ячеек) в строковом формате, куда нужно добавить выпадающий список;
- **dropdownRange** (string) — диапазон ячеек, в которых хранятся значения для выпадающего списка:
  - `A1:A10` — в выпадающем списке будут значения из листа, на котором будет находиться выпадающий список;
  - `'Лист3'!$A$1:$A$10` — значения будут браться из другого листа.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного создания выпадающего списка.

Пример

```
const result = await craftTechApi.createDropdownCell(fileHandler, 'Лист1',
'D20', `'Данные'!$C$3:$C$20`);
console.log('Выпадающий список добавлен: ', result);
```

createListBox

```
await craftTechApi.createListBox(array, sheetStr, start, defName = '', oColor =
Api.CreateColorFromRGB(0, 128, 0), isMultiple = false);
```

Описание

Формирует поле списка на листе на основе переданного диапазона. **Для создания выпадающего списка используйте *createDropdownCell***

> Данный метод был разработан ввиду ограничений Р7-Офис Таблицы, не предусматривающих создание и взаимодействие с пользовательскими формами и элементами управления. Единственный выход, который мы можем предложить заказчикам — явно имитировать действие поля списка через диапазон ячеек.

> Макрос, использующий данный метод, должен работать в режиме автозапуска (А).

Параметры

- **array** (string) — массив строк с опциями поля списка;
- **sheetStr** (string) — название листа, в который будет добавлено поле списка;
- **start** (string) — адрес ячейки, с которой начинается формирование поля списка;
- **defName** (string) — необязательный параметр; название поля списка как именованного диапазона (это необходимо, чтобы потом мы могли удобно получить выбранные опции из поля списка через метод getListBox); если название именованного диапазона не задано, макрос установит ему стандартное название listBox{i};
- **oColor** (ApiColor) — необязательный параметр; цвет ячейки, в которую она окрасится после клика на её при выборе опции; по умолчанию цвет `#008000`;
- **isMultiple** (boolean) — необязательный параметр; возможность множественного выбора опций в поле списка; по умолчанию отключено.

Возвращает

Метод ничего не возвращает.

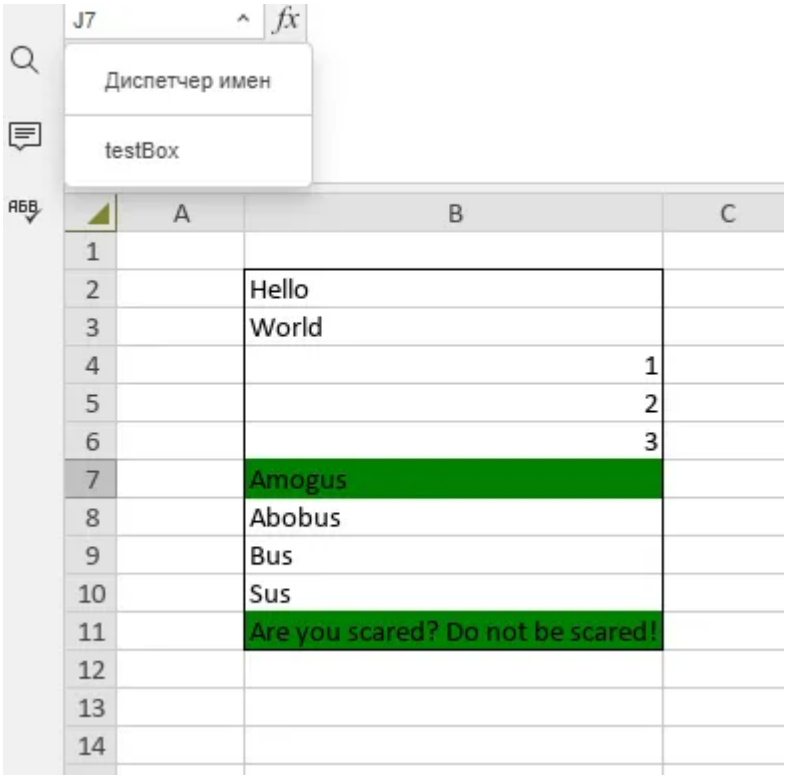
Пример

```
const array = ['Hello', 'World', '1', '2', '3', 'Amogus', 'Abobus', 'Bus',
'Sus', 'Are you scared? Do not be scared!'];
const defName = 'testBox';

function checkArrayType(input) {
  const isArray = Array.isArray(input);
  const isArrayOfArrays = isArray && input.every(item => Array.isArray(item));

  return { isArray, isArrayOfArrays };
}

await craftTechApi.createListBox(array, 'Лист2', 'B2', defName, '', true);
```



Код выше создаст поле списка, обведённое границами чёрного цвета. Каждый элемент массива, переданного первым аргументом, записывается в соответствующую ячейку как опция поля списка. Список в данном случае носит название «testBox».

В поле выбраны опции со значениями «Amogus» и «Are you scared? Do not be scared!» — они окрашены в зелёный цвет. Чтобы отменить опцию, необходимо ещё раз нажать на неё, чтобы ячейка стала бесцветной. Данное поле будет видно только на листе «Лист2».

createSheet

```
await craftTechApi.createSheet(fileHandler, sheetName);
```

Описание

Создаёт новый лист в книге.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для текущего файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа; необязательный параметр: в случае его отсутствия будет создан лист с названием «Лист 1».

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае успешного создания нового листа, иначе `false`.

Пример

```
const fh = await craftTechApi.openFile('book.xlsx');
await craftTechApi.createSheet(fh, 'newSheet');
await craftTechApi.saveFile(fh);
```

createStyledTable

```
await craftTechApi.createStyledTable(fileHandler, sheetName, styledDictConfig);
```

Описание

Создаёт стилизованную таблицу на листе

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **styledDictConfig** (object) — объект, который имеет поля:
  - **displayName** (string) — название стилизованной таблицы;
  - **ref** (string) — диапазон ячеек (например: `A1:C2`);
  - **tableStyleInfo**:
    - **name** (string) — название стиля таблицы;

      > Возможные значения `tableStyleInfo.name`: “TableStyleLight1” – “TableStyleLight21”, “TableStyleMedium1” – “TableStyleMedium28”, “TableStyleDark1” – “TableStyleDark11”

    - **showFirstColumn** (bool) — показывать ли первую колонку таблицы;
    - **showLastColumn** (bool) — показывать ли последнюю колонку таблицы;
    - **showRowStripes** (bool) — показывать ли разделители между рядами;
    - **showColumnStripes** (bool) — показывать ли разделители между колонками;

Возвращает

- **true / false** (boolean) — возвращает логическую истину в случае успешного выполнения метода.

Пример

```
var handler = await craftTechApi.openFile("book.xlsx")
let styledDictConfig = {
          tableStyleInfo: {
            name: 'TableStyleMedium2',
            showFirstColumn: 1,
            showLastColumn: 0,
            showRowStripes: 1,
            showColumnStripes: 0,
          },
          ref: 'B4:E11',
          displayName: 'styled table',
        }
await craftTechApi.createStyledTable(handler, 'Лист1', styledDictConfig);
```

createXlsx

```
await craftTechApi.createXlsx(filePath, sheetName);
```

Описание

Создаёт xlsx-файл по указанному пути с заданным названием листа. Если название не передано, то лист будет назван по умолчанию «Лист1».

Параметры

- **filePath** (string) — путь до создаваемого файла;
- **sheetName** (string) — название листа; необязательный параметр, если значение не передано, то лист будет назван «Лист1».

Возвращает

- **fileHandler** (number | boolean) — ссылка на новый файл, если файл создан успешно, иначе возвращает `false` в случае ошибки.

Пример

Создаём файл `Table.xlsx` с листом «HolySheet»:

```
const fileID = await craftTechApi.createXlsx('Table.xlsx', 'HolySheet');
if (fileID) {
  console.log('Файл создан с id:', fileID);
} else {
  console.log('При создании пустого xlsx произошла ошибка.');
}
```

Если мы хотим создать пустой xlsx с листом «Лист1», то пишем так:

```
const fileID = await craftTechApi.createXlsx('Table.xlsx');
```

decryptFile

```
await craftTechApi.decryptFile(filePath, password);
```

Описание

Снимает шифрование с файла. Файл вновь станет доступен для просмотра.

Параметры

- **filePath** (string) — путь до файла;
- **password** (string) — пароль для снятия защиты.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного дешифрования файла.

Пример

```
const isDecrypted = await craftTechApi.decryptFile('ivan.xlsx', 'qwerty');
console.log('Файл расшифрован: ', isDecrypted);
```

После выполнения данного кода с файла `ivan.xlsx` будет снят пароль (при условии, что пароль `qwerty` является правильным).

deleteActiveFileDrawings

```
await craftTechApi.deleteActiveFileDrawings(sheetName);
```

Описание

Удаляет рисунки (фигуры, графики, картинки) с листа активной книги.

> Требуется, чтобы лист НЕ был скрытым.

Параметры

- **sheetName** (string) — название листа.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного удаления фигур с листа.

Пример

```
const sheetsArr = ['Лист1', 'Лист2', 'Лист3'];
for (const sheet of sheetsArr) {
  const isDrawingsDeleted = await craftTechApi.deleteActiveFileDrawings(sheet);
  console.log(`На листе ${sheet} все рисунки удалены: ${isDrawingsDeleted}`);
}
```

deleteDrawings

```
await craftTechApi.deleteDrawings(fileName, sheetName, flag);
```

Описание

Удаляет рисунки(фигуры) с листа.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа.
- **flag** (string) (необязательный) — флаг, указывающий на то, какие элементы нужно удалить. Флаги можно перечислять через запятую, если необходимо удалить несколько типов (например, `button, chart`). Если вы не передаете флаг, то будут удалены все фигуры на листе. Доступны следующие флаги:
  - button - удаляет кнопки (shape с макросом)
  - shape - удаляет фигуры (но не кнопки)
  - chart - удаляет графики
  - picture - удаляет изображения

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного удаления фигур с листа.

Пример

```
const fileHandler = await craftTechApi.openFile('ivan.xlsx');
const isDeleted = await craftTechApi.deleteDrawings(fileHandler, 'Лист1', 'button, shape');
console.log('Шейпы удалены: ', isDeleted);
await craftTechApi.saveFile(fileHandler);
```

Пусть у листа «Лист1» файла `ivan.xlsx` есть документ, который нужно распечатать, и над ним на листе есть кнопки (фигуры, прямоугольники, линии и т.д.), которые запускают определённые макросы. После выполнения данного кода эти кнопки, как и любые другие фигуры (за исключением графиков), будут удалены.

deleteFile

```
await craftTechApi.deleteFile(filePath);
```

Описание

Удаляет файл по указанному пути.

Параметры

- **filePath** (string) — путь до файла.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного удаления файла. Если произошла ошибка или файл не существует, возвращается `false`.

Пример

```
const isDeleted = await craftTechApi.deleteFile('test.txt');
console.log('Файл удалён: ', isDeleted);
```

deleteRowFile

```
await craftTechApi.deleteRowFile(fileHandler, sheetName, rowNumber);
```

Описание

Очищает данные всего ряда на листе.

> Не путать с методом dropRowFile, который удаляет ряд на листе со смещением следующих рядов вверх.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **rowNumber** (number) — номер строки.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного очищения строки.

Пример

```
const result = await craftTechApi.dropRowFile(fileHandler, 'Лист1', 3);
console.log('Строка очищена: ', result);
```

Код выше очистит третий ряд в листе «Лист1», оставив его пустым.

deleteSheet

```
await craftTechApi.deleteSheet(fileHandler, sheetName);
```

Описание

Удаляет указанный лист в книге.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа, который необходимо удалить.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного удаления листа, иначе `false`.

Пример

```
const result = await craftTechApi.deleteSheet(fh, 'Лист2');
console.log('Лист удален: ', result);
```

deleteTempDirForce

```
craftTechApi.deleteTempDirForce();
```

Описание

Принудительно удаляет папку `temp`, в которой хранятся временные файлы, открытые через CraftTechPE, во время выполнения макроса.

Иногда бывает ситуация, что при возникновении какой-либо ошибки работа макроса прерывается досрочно, вследствие чего сессия и файлы, с которыми в данный момент взаимодействует макрос при помощи CraftTechPE, остаются открытыми, хотя по завершению работы макроса всё должно быть закрыто. При такой ситуации попытка повторно закрыть макрос может привести к непредсказуемым результатам, например, неактуальным данным. Данный метод позволит принудительно удалить папку `temp`, если CraftTechPE не смог это сделать автоматически.

Параметры

Метод не принимает параметров.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.deleteTempDirForce();
```

dropRowFile

```
await craftTechApi.dropRowFile(fileHandler, sheetName, rowNumber);
```

Описание

Удаляет данные всего ряда на листе со смещением вверх.

> Не путать с методом deleteRowFile, который просто очищает ряд на листе, оставляя его пустым.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **rowNumber** (number) — номер строки, либо диапазон строк (числа через двоеточие в виде строкового значения).

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного удаления строки.

Пример

```
const result = await craftTechApi.dropRowFile(fileHandler, 'Лист1', '3:6');
console.log('Строки удалены: ', result);
```

Код выше удалит с 3-го по 6-ой ряды на листе «Лист1», сместив находящиеся ниже ряды на три позиции вверх.

editCellRange

```
await craftTechApi.editCellRange(fileHandler, sheetName, cellAddress, data,
flag = 'value')
```

Описание

Вносит изменения в значения диапазона ячеек.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellAddress** (string) — диапазон ячеек;
- **data** (string | number | null)[][] | string — двумерный массив с числовыми или строковыми данными ячеек; если необходимо очистить значения диапазона ячеек, то передаётся пустая строка;
- **flag** (string) — необязательный параметр, указывающий, что именно пользователь хочет изменить:
  - value — в поле `data` передаются значения в виде строк (по умолчанию);
  - formula — флаг для изменения формулы ячейки, в поле `data` передаются формулы в виде строк;
  - value, formula — флаг для изменения значения и формулы, в поле `data` передаются объекты в следующем формате:

```
{
  value: 'Строка', // значение, которое будет заменено
  formula: 'C1+A5', // формула ячейки
}
```

Передаются данные в виде одного большого массива, а в нём — другие массивы, которые соответствуют порядковому номеру ряда в документе. Элементы внутри второго массива соответствуют порядку столбцов в введённом диапазоне. Если мы пишем ” (пустая строка) — это означает, что мы хотим пропустить ячейку.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного изменения ячеек.

Пример

```
const result = await craftTechApi.editCellRange(
  fileHandler,
  'Лист1',
  'F102:G103',
    [
      ['TEST', 'METHOD'],
      ['', 123],
    ]
  );

// очищение: const result = await craftTechApi.editCellRange(fileHandler, 'Лист1',
'F102:G103', '')
console.log('Значения изменены: ', result);
```

editCells

```
await craftTechApi.editCells(fileHandler, sheetName, data)
```

Описание

Изменяет значение ячейки и применяет форматирование к ней.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **data** (object[]) — массив объектов; каждый из этих объектов содержит данные о конкретной ячейке, включая адрес, данные и стили (одна ячейка === один объект).
    - Для установки стиля передаётся объект с соответствующими полями и их значениями; в объекте должны быть только те стили, которые мы хотим задать. Например, чтобы изменить размер текста, достаточно передать в style `{ font: {fontSize: 13} }`.
        - address (string) — передаётся адрес ячейки, которую хотим изменить;
        - value (string | number) — передаётся значение ячейки;
        - formula (object) — передается объект formula с полем formula, в котором хранится формула ячейки;
        - style (object) — конфигурация стилей ячейки.

      При этом остальные параметры стиля в ячейке вернутся к стандартным, так что при изменении или добавления параметра прописывайте все параметры которые были у ячейки!

Доступные конфигурации стилей:

СТИЛЬ ТЕКСТА (font: {})

- bold (boolean) – полужирный шрифт;
- italic (boolean) – шрифт курсивом;
- underline (boolean) – нижнее подчёркивание;
- fontSize (number) – размер шрифта (в pt);
- fontName (string) – название шрифта;
- colorRgb (string) – цвет текста (RGB);
- colorIndexed (number) – цвет текста (Индексированный);
- colorTheme (number) – цвет текста (Тема);
- colorTint (number) – акцент белого с цветом темы от 0 до 1 (Применяется только в связке с colorTheme).

ФОН (fill: {})

- patternType (string) - стиль паттерна, по умолчанию solid;
- fgRgb (string) - цвет переднего фона (RGB);
- fgIndexed (string) - цвет переднего фона (Индексированный);
- fgTheme (string) - цвет переднего фона (Тема);
- fgTint (string) - акцент белого с цветом темы от 0 до 1 (Применяется только в связке с fgTheme);
- bgRgb (string) - цвет заднего фона (RGB);
- bgIndexed (string) - цвет заднего фона (Индексированный);
- bgTheme (string) - цвет заднего фона (Тема);
- bgTint (string) - акцент белого с цветом темы от 0 до 1 (Применяется только в связке с bgTheme).

ГРАНИЦЫ (border)

- border: ‘string’ – шорт кат, позволяющий разом установить одни и те же стили для четырёх сторон ячейки (вверх, вниз, влево, вправо).

> Таким способом можно изменить **только** style(например, thin), для более детального изменения нужно прописывать свойства для каждой границы, как показано ниже.

СВОЙСТВА ГРАНИЦ (border: {})

Далее идет определение полей объекта границ, если вам требуется более гибкая настройка. У границ есть четыре направления(далее direction): left, right,

`top, bottom. Описанные свойства применяются исключительно с прописыванием стороны, к которой применяется изменение. Например, для применения ColorRgb к левой стороне границы, необходимо написать leftColorRgb.`

- {direction}Style - тип границы (например, thin);
- {direction}ColorRgb - цвет границы (RGB);
- {direction}ColorIndexed - цвет границы (Индексированный);
- {direction}ColorTheme - цвет границы (Тема);
- {direction}ColorTint - акцент белого с цветом темы от 0 до 1 (Применяется только в связке с ColorTheme).

Пример названия параметров: leftColorRgb, rightColorTint, topColorRgb, …

ВЫРАВНИВАНИЕ (alignment: {})

- alignmentHorizontal (string) [“left”, “right”, “center”] – устанавливает горизонтальное выравнивание текста;
- alignmentVertical (string) [“top”, “bottom”, “center”] – устанавливает вертикальное выравнивание текста;
- alignmentWrapText (boolean) - указывает, будут ли слова в ячейке переноситься в соответствии с размером ячейки;
- textRotation (number) - количество градусов, на которое необходимо повернуть текст.

~~*ФОРМАТИРОВАНИЕ ДАННЫХ* - numberFormat (string) - устанавливает тип/маску, которой должно соответствовать число в ячейке.~~

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае успешного изменения ячеек.

Пример

```
const editedCells = await craftTechApi.editCells(
  fileHandler,
  'Лист2',
  [
    {
      address: 'A1',
      value: 'hello',
      formula: { formula: 'A13+A14' },
      style: {
        font: { bold: true, underline: true },
        border: { rightStyle: 'thin', rightColorRgb: 'FF000000' },
        fill: { bgRgb: 'FF2A8DD4', fgRgb: 'FF2A8DD4' }
      }
    },
    {
      address: 'B7',
      value: 'planet',
      style: {
        border: 'thin',
        font: { fontSize: 13, italic: true, bold: true }
      }
    },
    {
      address: 'C4',
      value: 'dog'
    }
  ]
);

console.log('Метод editCells вернул значение:', editedCells)
```

В данном случае:

- A1 – хранит значение “hello” с полужирным, подчёркнутым шрифтом и правая граница ячейки тонко окрашена, а также иметь заливку;
- B7 – хранит значение “planet” с полужирным шрифтом и курсивом, кегль 13pt, а границы ячейки тонко окрашены;
- C4 – хранит значение “dog” с прежними стилями (если они были);

editCellsValue

```
await craftTechApi.editCellsValue(fileHandler, sheetName, data)
```

Описание

Вносит изменения в значения нескольких ячеек.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **data** (string | number | null)[][] — двумерный массив с числовыми или строковыми данными ячеек.

Данные ячеек передаются аналогично получаемым данным в методе **getSheetArray**. То есть всё кладём в один большой массив, а в нём другие массивы, которые соответствуют порядковому номеру ряда в документе. Элементы внутри второго массива соответствуют порядку столбцов в документе. Если мы пишем null — это означает, что мы хотим пропустить ряд или столбец.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного изменения ячеек.

Пример

```
const result = await craftTechApi.editCellsValue(
  fileHandler,
  'Лист1', [
    ['Я ячейка A1', 'A я B1'],
    null,
    ['Я A3', null, 'A тут C3'],
  ]
);

console.log('Значения добавлены: ', result);
```

Пусть мы имеем таблицу следующего формата:

```
A1 - B1 - C1
A2 - B2 - C2
A3 - B3 - C3
```

Тогда, после вызова метода из примера выше мы изменили значения ячеек, которые отмечены (*****), остальные ячейки остались нетронутыми:

```
A1* - B1* - C1
A2  - B2  - C2
A3* - B3  - C3*
```

editCellValue

```
await craftTechApi.editCellValue(fileHandler, sheetName, cellAddress, data,
flag = 'value')
```

Описание

Вносит изменения в значение ячейки.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellAddress** (string) — адрес ячейки в строковом формате (например: `C1`, либо `1,3`; во втором случае сначала передаётся строка, а потом столбец);
- **flag** (string) — необязательный параметр, указывающий, что именно пользователь хочет изменить:
  - value — в поле `value` передается значение в формате строки (по умолчанию);
  - formula — флаг для изменения формулы ячейки, в поле `value` передается значение в формате строки;
  - value, formula — флаг для изменения значения и формулы, в поле `value` передается объект в следующем формате:

```
let value = {
  value: 'Строка', // значение, которое будет заменено
  formula: 'C1+A5', // формула ячейки
}
```

Возвращает

- **true | false** (boolean) — возвращает логическую истину в случае успешного изменения ячейки.

Пример

```
const result = await craftTechApi.editCellValue(fileHandler, 'Лист1', '2,4',
'777');

if (result) {
  console.log('Значение добавлено в ячейку B4');
} else {
  console.log('Возникла ошибка');
}
```

editSheetPrintOptions

```
await craftTechApi.editSheetPrintOptions(fileHandler, sheetName, config);
```

Описание

Изменяет параметры печати листа.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **config** (object) — объект с полями, которые необходимо поменять:
  - orientation (string) — ориентация листа (`portrait`, `landscape`);
  - zoom (number) — размер увеличения на листе;
  - fitToPage (boolean) — подгонка размера под содержимое (`true` или `false`);
  - marginLeft (number) — отступ слева;
  - marginRight (number) — отступ справа;
  - marginTop (number) — отступ сверху;
  - marginBottom (number) — отступ снизу;
  - horizontalCentered (number) — центровка по горизонтали (1 или 0);
  - paperSize (number) — размер бумаги (см. Размеры бумаги в Р7-Офис ниже).
  - printArea (string) — диапазон ячеек на листе, которые будут входить в область печати.

Размеры бумаги в Р7-Офис

Все размеры указаны в сантиметрах (см)

1. **US Letter** (21.59 x 27.94)
2. **US Legal** (21.59 x 35.56)

3. **A4** (21 x 29.7)
4. **A5** (14.8 x 21)
5. **B5** (17.6 x 25)
6. **Envelope 10** (10.48 x 24.13)
7. **Envelope DL** (11 x 22)
8. **Tabloid** (27.94 x 43.18)
9. **A3** (29.7 x 42)
10. **Tabloid Oversize** (30.48 x 45.71)
11. **ROC 16K** (19.68 x 27.3)
12. **Envelope Choukei 3** (11.99 x 23.49)
13. **SuperB/A3** (33.02 x 48.25)

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае успешного изменения параметров печати листа, иначе `false`.

Пример

```
const conf = {
   orientation: 'portrait',
   zoom: 1.02,
   fitToPage: true,
   marginLeft: 0.77,
   marginRight: 0.77,
   marginTop: 1,
   marginBottom: 1,
   horizontalCentered: 1,
   paperSize: 5,
   printArea: 'A1:B10',
}

const result = await craftTechApi.editSheetPrintOptions(fileHandler, 'Лист1',
conf);
console.log('Параметры изменены: ', result);
```

editSheetTabColor

```
await craftTechApi.editSheetTabColor(fileHandler, sheetName, color);
```

Описание

Изменяет цвет вкладки листа в документе.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для текущего файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **color** (string) — цвет в формате HEX с префиксом «FF» (например, если нужен цвет с кодом `#7351DB`, значит должна передаваться строка «FF7351DB».

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае успешной перекраски вкладки листа, иначе `false`.

Пример

```
const isChanged = await craftTechApi.editSheetTabColor(fileHandler, 'Лист1',
'FF7351DB');
console.log('Цвет вкладки изменён: ', isChanged);
```

editSheetVisibility

```
await craftTechApi.editSheetVisibility(fileHandler, sheetName, state)
```

Описание

Изменяет видимость листа.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **state** (string) — состояние листа (hidden/visible);

Возвращает

- **true | false** (boolean) — возвращает логическую истину в случае успешного изменения состояние листа.

Пример

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
await craftTechApi.editSheetVisibility(fileHandler, 'Лист1', 'visible')
```

editTxtFile

```
await craftTechApi.editTxtFile(oldFilePath, startRow, data, newFilePath);
```

Описание

Вносит изменения в текстовый файл и сохраняет его под новым названием.

Параметры

- **oldFilePath** (string) — путь до файла источника (необязательный параметр, по умолчанию пустая строка);
- **startRow** (number) — номер строки, начиная с нуля, с которой мы вставим данные;
- **data** (array) — массив данных, каждый элемент которого представляет собой отдельную строку в текстовом файле;
- **newFilePath** (string) — путь до нового файла.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае изменения текстового файла, иначе `false`.

Пример

Если первым параметром передается пустая строка, метод создаст новый файл и будет вносить изменения в него, не основываясь на другом файле. Метод перезаписывает строки в документе теми данными, которые мы передали.

Мы хотим внести изменения в файл `old.txt`, а именно перезаписать данные начиная с 3-й строки:

- 3-я строка: «я»;
- 4-я строка: «люблю»;
- 5-я строка: «макросы».

Затем сохранить этот файл под новым названием `new.txt`.

```
await craftTechApi.editTxtFile(
  'old.txt',
  2,
  ['я', 'люблю', 'макросы'],
  'new.txt'
);
```

Другой пример. Мы хотим создать новый текстовый файл и написать на первой строке «Привет», а на пятой «Крафттек», вызовем метод:

```
await craftTechApi.editTxtFile(
  '',
  0,
  ['Привет', '', '', '', 'Крафттек'],
  '/home/ivan/files/new.txt'
);
```

Первым параметром передаем пустую строку, так как хотим создать новый файл. Вторым параметром передаем `0`, так как начинаем с первой строки. Третий параметр — массив с 5 элементами, которые будут соответствовать порядковым номерам строк в новом текстовом файле. Четвертым параметром указываем путь до создаваемого файла.

encryptFile

```
await craftTechApi.encryptFile(filePath, password);
```

Описание

Шифрует файл. Файл станет недоступен для просмотра, и при попытке его открыть потребуется ввод пароля пользователем.

Параметры

- **filePath** (string) — путь до файла;
- **password** (string) — пароль для защиты.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного шифрования файла.

Пример

```
const isEncrypted = await craftTechApi.encryptFile('ivan.xlsx', 'qwerty');
console.log('Файл зашифрован: ', isEncrypted);
```

После выполнения данного кода на файл `ivan.xlsx` будет установлен пароль.

executeMacros

```
await craftTechApi.executeMacros(filePath, macroPath);
```

Описание

Запускает макрос внешнего файла.

Параметры

- **filePath** (string) — путь до внешнего файла `.xlsx`;
- **macrosPath** (string) — путь до файла с макросом с расширением `.js`/`.txt`.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного запуска макроса.

Пример

```
const result = await craftTechApi.executeMacros('book.xlsx', 'macro.js');
console.log('Макрос запущен: ', result);
```

existDir

```
await craftTechApi.existDir(dirPath);
```

Описание

Проверяет существование папки по указанному пути.

Параметры

- **dirPath** (string) — путь до папки.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае, если папка существует, иначе `false`.

Пример

```
const isExist = await craftTechApi.existDir('/files/work');
console.log('Папка существует: ', isExist);
```

existFile

```
await craftTechApi.existFile(path);
```

Описание

Проверяет существование файла по указанному пути.

Параметры

- **path** (string) — путь до файла.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае, если файл существует, иначе `false`.

Пример

```
const isExist = await craftTechApi.existFile('book.xlsx');
console.log('Файл существует: ', isExist);
```

findCellsBySubString

```
await craftTechApi.findCellsBySubString(fileHandler, sheetName, value, flag)
```

Описание

Находит все ячейки листа, или диапазона, содержащие в своём значении подстроку.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **value** (string) — значение, которому должна соответствовать ячейка.
- **flag** (string/object) (Необязательный параметр) - строковое значение в котором указан диапазон ‘A1:C5’, или объект с ключами {range: ‘A1:C5’, resType: ‘new’}. При этом в объекте может присутствовать и только один из ключей. Если есть ключ range - то поиск будет происходить в конкретном диапазоне, если есть ключ resType - то будет изменен тип возвращаемой информации.

Возвращает

Возврат без ключа resType (Поведение по умолчанию)

- **addressRange** (string[]) — массив строк с адресами найденных ячеек.

Возврат с ключом resType

- **addressRangeNew** (object[]) - массив объектов с адресами, номерами строк и колонок и значением ячейки. Пример [{address: ‘A1’, col: 1, row: 1, value: ‘amogus’},{address: ‘C2’, col: 3, row: 2, value: ‘amogus’}, {address: ‘A3’, col: 1, row: 3, value: ‘gugus’}]. Значения address и value - типа string, а col и row - типа number

Пример

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const range = await craftTechApi.findCellsByValue(fileHandler, 'Лист1', 'gus');
console.log('Подстрока gus найдена в следующих ячейках: ', range);
```

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const range = await craftTechApi.findCellsByValue(fileHandler, 'Лист1', 'gus',
{range: 'A15:D20', resType: 'new'});
console.log('Подстрока gus найдена в следующих ячейках: ', range);
```

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const range = await craftTechApi.findCellsByValue(fileHandler, 'Лист1', 'gus',
'A15:D20');
console.log('Подстрока gus найдена в следующих ячейках: ', range);
```

findCellsByValue

```
await craftTechApi.findCellsByValue(fileHandler, sheetName, value, flag)
```

Описание

Находит все ячейки листа, соответствующие указанному значению.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **value** (string) — значение, которому должна соответствовать ячейка.
- **flag** (string/object) (Необязательный параметр) - строковое значение в котором указан диапазон ‘A1:C5’, или объект с ключами {range: ‘A1:C5’, resType: ‘new’}. При этом в объекте может присутствовать и только один из ключей. Если есть ключ range - то поиск будет происходить в конкретном диапазоне, если есть ключ resType - то будет будет изменен тип возвращаемой информации.

Возвращает

Возврат без ключа resType (Поведение по умолчанию)

- **addressRange** (string[]) — массив строк с адресами найденных ячеек.
- **Возврат с ключом resType**
- **addressRangeNew** (object[]) - массив объектов с адресами, номерами строк и колонок и значением ячейки. Пример [{address: ‘A1’, col: 1, row: 1, value: ‘amogus’},{address: ‘C2’, col: 3, row: 2, value: ‘amogus’}]. Значения address и value - типа string, a col и row - типа number

Пример

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const range = await craftTechApi.findCellsByValue(fileHandler, 'Лист1', 'amogus');
console.log('amogus найден в следующих ячейках: ', range);
```

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const range = await craftTechApi.findCellsByValue(fileHandler, 'Лист1', 'amogus',
{range: 'A15:D20', resType: 'new'});
console.log('amogus найден в следующих ячейках: ', range);
```

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const range = await craftTechApi.findCellsByValue(fileHandler, 'Лист1', 'amogus',
'A15:D20');
console.log('amogus найден в следующих ячейках: ', range);
```

freezeField

```
await craftTechApi.freezeField(fileHandler, sheetName, number, flag = 'null');
```

Описание

Закрепляет строки и столбцы.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **number** (string) — номер строки или столбца (в зависимости от значения параметра flag) для закрепления;
- **flag** (string) — флаг для уточнения координаты:
  - all — и по строке, и по столбцу;
  - row — по строке;
  - column — по столбцу.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного закрепления ячеек.

Пример

```
const freeze = await craftTechApi.freezeField(fileHandler, 'Лист1', 'row');
console.log('Ряд закреплён: ', freeze);
```

getActiveSheetName

```
await craftTechApi.getActiveSheetName(fileHandler);
```

Описание

Возвращает активный лист переданного файла.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.

Возвращает

- **sheetName** (string) — название активного листа.

Пример

```
const result = await craftTechApi.getActiveSheet(fileHandler);
console.log('Активный лист: ' result);
```

Пусть есть файл `book.xlsx` с листами «Лист1», «Отчёт» и «Декларация». Пользователь проделал какие-то действия на листе «Отчёт», сохранил и закрыл этот файл.

Код выше вернёт название листа, которое было активным при последнем действии пользователя, то есть «Отчёт».

getAllCharts

```
await craftTechApi.getAllCharts(fileHandler);
```

Описание

Возвращает все графики переданного файла.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.

Возвращает

- **chartsObj** (object[]) — массив объектов с графиками в следующем формате:
  - position — порядковый номер графика на листе;
  - name — название графика, взятое из Drawing;
  - title — заголовок графика;
  - sheetName — имя листа, на котором находится график;
  - chart — информация о графике;
    - cat — категория графика;
    - legends — легенды графика;
      - name — ячейка, из которой берется название легенды;
      - value — диапазон значений для графика.

```
{
   position: 1, // порядковый номер графика на листе
   name: 'Chart 10', // название графика, взятое из Drawing
   title: 'Заголовок графика', // заголовок графика
   sheetName: 'Лист1', // имя листа, на котором находится график
   chart: { // информация о графике
       cat: "'Для графиков'!$AC$1:$AE$1", // категория графика
       legends: [ // легенды графика
           {
               name: "'Для графиков'!$AB$16", // ячейка, из которой берется
название легенды
               value: "'Для графиков'!$AC$16:$AE$16", // диапазон значений для
графика
           },
           {
                name: "'Для графиков'!$AB$15",
                value: "[0]!Лист10",
           }
       ]
   }
} // В будущем данный метод будет возвращать больше данных.
```

Пример

```
const allCharts = await craftTechApi.getAllCharts(fileHandler);
console.log('Все графики: ', allCharts);
```

getAllChartsBySheet

```
await craftTechApi.getAllChartsBySheet(fileHandler, sheetName);
```

Описание

Возвращает все графики указанного листа переданного файла.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа.

Возвращает

- **chartsObj** (object[]) — массив объектов с графиками в следующем формате:
  - position — порядковый номер графика на листе;
  - name — название графика, взятое из Drawing;
  - title — заголовок графика;
  - sheetName — имя листа, на котором находится график;
  - chart — информация о графике;
    - cat — категория графика;
    - legends — легенды графика;
      - name — ячейка, из которой берется название легенды;
      - value — диапазон значений для графика.

```
{
   position: 1, // порядковый номер графика на листе
   name: 'Chart 10', // название графика, взятое из Drawing
   title: 'Заголовок графика', // заголовок графика
   sheetName: 'Лист1', // имя листа, на котором находится график
   chart: { // информация о графике
       cat: "'Для графиков'!$AC$1:$AE$1", // категория графика
       legends: [ // легенды графика
           {
               name: "'Для графиков'!$AB$16", // ячейка, из которой берется
название легенды
               value: "'Для графиков'!$AC$16:$AE$16", // диапазон значений для
графика
           },
           {
                name: "'Для графиков'!$AB$15",
                value: "[0]!Лист10",
           }
       ]
   }
} // В будущем данный метод будет возвращать больше данных.
```

Пример

```
const allCharts = await craftTechApi.getAllChartsBySheet(fileHandler, 'Лист1');
console.log('Все графики Листа1: ', allCharts);
```

getBaseDir

```
await craftTechApi.getBaseDir();
```

Описание

Позволяет получить путь до папки, внутри которой находится активный файл (откуда запускается макрос).

> Это асинхронная функция. Для выполнения действия синхронно используйте `getBaseDirSync` — он работает аналогичным образом.

Параметры

Метод не принимает параметров.

Возвращает

- **folderPath** (string) — абсолютный путь до папки с файлом, из которого запущен текущий макрос.

Пример

```
const result = await craftTechApi.getBaseDir();
console.log(result); // тут путь до папки
```

getCellRange

```
await craftTechApi.getCellRange(fileHandler, sheetName, cellRange, flag)
```

Описание

Возвращает значения диапазона ячеек.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cell** (string) — диапазон ячеек в строковом формате, адреса ячеек разделяются двоеточием (например: `A1:C2`, либо `1,1:2,3`; во втором случае сначала передаётся строка, а потом столбец);
- **flag** (string | null) — флаг, указывающий, какие данные необходимо получить; по умолчанию `null`, возвращает стили, значения и адреса; доступны следующие флаги: `style` - возвращает стили, `formula` - возвращает формулы, `sizes` - возвращает размеры ячеек. Флаги необходимо указывать через запятую. Например, если вы хотите получить стили и формулы, необходимо написать `style, formula`.

Возвращает

- **cellRange** (object[][]) — двумерный массив значений диапазона ячеек; данные ячеек представляют собой объект, в котором строковое или числовое значение ячейки хранится под ключом `value`, стиль ячейки со всеми полями хранится под ключом `style`, а также объект с ключом `address`, в котором хранятся три поля:
  - **cell** (string) — строковое представление адреса;
  - **col** (number) — числовое значение колонки;
  - **row** (number) — числовое значение ряда.

  Помимо этого, при передаче флага `sizes`, вы получаете в объекте ячейки поле `size`, в котором хранятся два ключа:

- **height** (number) — высота строки ячейки в пунктах;
- **width** (number) — ширина столбца ячейки в символах;

Пример

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const range = await craftTechApi.getCellRange(fileHandler, 'Лист1', 'A1:C3');
console.log('Содержимое ячеек A1:C3: ', range);
```

Вывод:

```
[
 [
   { value: 'Word', style: { font: { bold: true } }, address: { cell: 'A1', col: 1,
row: 1 } },
   { value: 'Time', style: {}, address: { cell: 'B1', col: 2, row: 1 } },
   { value: 'Number', style: {}, address: { cell: 'C1', col: 3, row: 1 } }
 ],
 [
   { value: 'Water', style: {}, address: { cell: 'A2', col: 1, row: 2 } },
   { value: 'Day', style: {}, address: { cell: 'B2', col: 2, row: 2 } },
   { value: 'Part', style: {}, address: { cell: 'C2', col: 3, row: 2 } }
 ],
 [
   { value: 'Place', style: {}, address: { cell: 'A3', col: 1, row: 3 } },
   { value: 4444, style: {}, address: { cell: 'B3', col: 2, row: 3 } },
   { value: 'Back', style: {}, address: { cell: 'C3', col: 3, row: 3 } }
 ]
]
```

В данном случае массив содержит в себе данные трёх рядов из заданного диапазона на листе «Лист1». Порядок рядов хранится сверху-вниз. Данные ячеек хранятся слева-направо: первое значение — это самый левый ряд.

getCellValue

```
await craftTechApi.getCellValue(fileHandler, sheetName, cell)
```

Описание

Возвращает значение ячейки.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cell** (string) — адрес ячейки в строковом формате (например: `C1`, либо `1,3`; во втором случае сначала передаётся строка, а потом столбец).

Возвращает

- **cellValue** (string | number | null) — значение ячейки; если ячейка не хранит в себе значений, то вернётся `null`.

Пример

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const cell = await craftTechApi.getCellValue(fileHandler, 'Лист1', 'D7');
// const cell = await craftTechApi.getCellValue(fileHandler, 'Лист1', '7,4') //
аналогичная запись
console.log('содержимое ячейки D7: ', cell, typeof(cell));
```

getColumnIndex

```
await craftTechApi.getColumnIndex(cellAddress);
```

Описание

Возвращает номер колонки, в которой находится ячейка на активном листе.

Параметры

- **cellAddress** (string) — адрес ячейки.

Возвращает

- **columnIndex** (number) — номер колонки.

Пример

```
const result = await craftTechApi.getColumnIndex('C20');
console.log(result); // вернёт 3
```

getCurrentFilePath

```
await craftTechApi.getCurrentFilePath();
```

Описание

Позволяет получить путь до активного файла (откуда запускается макрос).

> Это асинхронная функция. Для выполнения действия синхронно используйте `getCurrentFilePathSync` — он работает аналогичным образом.

Параметры

Метод не принимает параметров.

Возвращает

- **filePath** (string) — абсолютный путь до файла, из которого запущен текущий макрос.

Пример

```
const result = await craftTechApi.getCurrentFilePath();
console.log(result); // тут путь до файла
```

getDefaultCheckboxValues

```
craftTechApi.getDefaultCheckboxValues()
```

Описание

Возвращает символы чекбоксов.

Параметры

Метод не принимает параметров.

Возвращает

Массив символов.

Пример

```
const [yes, no] = craftTechApi.getDefaultCheckboxValues();
```

getDefaultRadioboxValues

```
craftTechApi.getDefaultRadioboxValues()
```

Описание

Радиобоксы работают на символах, чтобы не получать эти символы из юникода, создан данный метод, который возвращает массив значений [yes, no].

Параметры

Метод не принимает параметров.

Возвращает

Массив символов.

Пример

```
const [yes, no] = craftTechApi.getDefaultRadioboxValues();
```

getDirPathsByDialog

```
await craftTechApi.getDirPathsByDialog();
```

Описание

Открывает диалоговое окно в Проводнике и возвращает абсолютный путь до папки, выбранной пользователем в этом окне.

Параметры

Метод не принимает параметров.

Возвращает

- **filePath** (string) — строка с абсолютным путём до выбранной папки.

Пример

```
const folder = await craftTechApi.getDirPathsByDialog();
console.log(folder);
```

getDocTables

```
await craftTechApi.getDocTables(fileHandler);
```

Описание

Возвращает содержимое из таблиц электронного документа.

Параметры

- **fileHandler** (number) — ссылка на файл.

Возвращает

- **docTables** (object[]) — содержимое из таблиц электронного документа. Метод возвращает массив объектов, где каждый объект — это информация по каждой таблице электронного документа. Каждый такой объект содержит следующие поля (возможно, в будущем будут новые поля):
  - content (string[][]) — содержимое таблицы в виде двумерного массива со строковыми значениями;
  - index (number) — индекс таблицы в документе, начиная с нуля.

Пример

```
const handler = await craftTechApi.openFile('crafttech.docx');
const result = await craftTechApi.getDocTables(handler);
console.log('Все таблицы: ', result);
```

getFieldLength

```
await craftTechApi.getFieldLength(fileHandler, sheetName, number, flag);
```

Описание

Находит длину переданного ряда или столбца.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **number** (string) — номер строки или столбца (в зависимости от значения flag);
- **flag** (string) — флаг, указывающий, где проводить поиск:
  - row — поиск в строках;
  - column — поиск в столбцах.

Возвращает

- **fieldLength** (number) — число, соответствующее длине переданного ряда или столбца.

Пример

```
const res = await craftTechApi.getFieldLength(fileHandler, 'Лист1', 3, 'row');
console.log(res);
```

Пусть имеется таблица вида:

```
[
  [1, 2, 3, 4, 5],
  [6, 7, 8, 9, 10],
  [11, 12, 13, 14, 15, 16],
  [17, 18, 19, 20, 21],
  [22, 23, 24, 25, 26]
];
```

Тогда результатом кода, представленного выше, будет 6.

getFileData

```
await craftTechApi.getFileData(fileHandler)
```

Описание

Возвращает все текстовые и числовые данные в документе.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.

Возвращает

- **fileData** (object) — объект с данными по документу, ссылку на которую передали параметром; ключ соответствует названию листа, а значением является массив, который содержит все строковые и числовые данные ячеек на листе; данные ячейки хранятся в виде объекта, где ключ соответствует адресу ячейки, а значение соответствует значению ячейки (строковому или числовому).

```
{
  'Лист1': [
    {A1: 'Кошки'},
    {A4: 'Собаки'}
  ],
  'Лист2': [
    {A1: 'Птицы'},
    {B1: 321}
  ]
}
```

Здесь мы видим, что:

1. Файл содержит 2 листа с названиями «Лист1» и «Лист2»;
2. Только две ячейки на листе «Лист1» содержат данные — это A1 и A4.

Пример

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const data = await craftTechApi.getFileData(fileHandler);
console.log('Содержание файла: ', data);
```

getFilesInDir

```
await craftTechApi.getFilesInDir(dirName, depth = 0);
```

Описание

Используется для получения абсолютных путей до файлов, которые находятся в указанной папке.

Параметры

- **dirName** (string) — путь до папки, внутри которой необходимо проводить поиск;
- **depth** (string) — глубина поиска, начиная с 1; если передать в параметр 0 (по умолчанию), то будет проведен поиск папок и файлов по всем возможным папкам и подпапкам.

Возвращает

- **filesInDir** (string[][]) — массив с массивами, в которых первый элемент — это полный путь до файла, но без самого названия файла, а второй элемент — это название файла.

Пример

```
const paths = await craftTechApi.getFilesInDir('/home/ivan/project/cart', 1);
```

Пусть мы хотим получить все пути до файлов в директории cart со следующей структурой:

```
.
└─cart

  ├─fruits
    ├─apple.txt
    └─grape.js

  ├─meat
    ├─dietary
      ├─chicken.txt
      └─turkey.js
    ├─cow.js
    └─pork.php

  └─emptyDir
```

Тогда, вызывая метод `getFilesInDir()`, мы получаем в ответ объект:

```
[
  ['/home/ivan/project/cart/fruits/', 'apple.txt'],
  ['/home/ivan/project/cart/fruits/', 'grape.js'],
  ['/home/ivan/project/cart/meat/', 'cow.js'],
  ['/home/ivan/project/cart/meat/', 'pork.php'],
  ['/home/ivan/project/cart/meat/dietary/', 'chicken.txt'],
  ['/home/ivan/project/cart/meat/dietary/', 'turkey.js'],
  ['/home/ivan/project/cart/emptyDir', '']
]
```

В данном примере мы получили пути до всех шести файлов, находящихся в директории. Первым элементом каждого массива является полный путь до файла, вторым элементом является название файла. Если директория пустая, то вторым элементом в полученных данных будет пустая строка.

Если мы хотим получить пути к файлам и папкам в текущей директории, то передаём методу точку:

```
const paths = await craftTechApi.getFilesInDir('.');
```

Если же мы хотим получить пути к файлам только в текущей директории, то передаем точку и 1:

```
const paths = await craftTechApi.getFilesInDir('.', 1);
```

getHtml

```
await craftTechApi.getHtml(link, elem_id = 'null');
```

Описание

Возвращает HTML-разметку переданного URL-адреса.

Параметры

- **link** (string) — URL-адрес ресурса;
- **elem_id** (string) — необязательный параметр; позволяет получить конкретный элемент и его дочерние элементы через уникальный идентификатор (атрибут `id`), символ # указывать **не нужно**; если параметр не задан, Вы получите корневой объект DOM, содержащий всё дерево элементов.

Возвращает

- **htmlString** (string) — элемент в строковом формате с сохранением табуляции.

Пример

```
const result = await craftTechApi.getHtml('https://crafttech.ru/');
console.log('DOM: ', result);
```

getListBox

```
await craftTechApi.getListBox(sheetStr, defName);
```

Описание

Возвращает данные, выбранные пользователем из поля списка.

> Данный метод был разработан ввиду ограничений Р7-Офис Таблицы, не предусматривающих создание и взаимодействие с пользовательскими формами и элементами управления. Единственный выход, который мы можем предложить заказчикам — явно имитировать действие поля списка через диапазон ячеек.

> Макрос, использующий данный метод, должен работать в режиме автозапуска (А).

Параметры

- **sheetStr** (string) — название листа, в котором находится поле списка;
- **defName** (string) — название поля списка как именованного диапазона.

Возвращает

- **listBoxData** (object) — объект с выбранными ячейками:
  - counter (number) — количество выбранных ячеек;
  - total (number) — общее количество ячеек в поле списка;
  - data (object[]) — массив объектов с информацией о выбранных ячейках:
    - address (string) — адрес выбранной ячейки;
    - cell (ApiRange) — объект ApiRange выбранной ячейки;
    - index (number) — индекс (порядок) выбранной ячейки в поле списка;
    - value (string) — строковое значение выбранной ячейки.

Пример

```
const result = await craftTechApi.getListBox('Лист2', 'testBox');
console.log(result);
```

Код выше получит информацию о выбранных ячейках поля списка, ранее созданного на странице, посвящённой методу createListBox().

Метод вернёт следующий результат:


```
{
  "counter": 2,
  "total": 10,
  "data": [
    {
      "address": "B7",
      "cell": ApiRange,
      "index": 5,
      "value": "Amogus"
    },
    {
      "address": "B11",
      "cell": ApiRange,
      "index": 9,
      "value": "Are you scared? Do not be scared!"
    }
  ]
}
```

getPageCount

```
await craftTechApi.getPageCount(filePath);
```

Описание

Возвращает количество листов переданного файла формата .doc и .docx.

Параметры

- **filePath** (string) — путь до файла.

Возвращает

- **pageCount** (number) — количество листов (страниц) переданного файла.

Пример

```
const result = await craftTechApi.getPageCount('crafttech.docx');
console.log('Количество листов файла crafttech.docx: ', result);
```

getPathsByDialog

```
await craftTechApi.getPathsByDialog(type, isMultiple, isWarn);
```

Описание

Открывает диалоговое окно в Проводнике и возвращает абсолютные пути до файлов, выбранных пользователем в этом окне.

Параметры

- **type** (string) — тип поддерживаемых файлов в строковом формате:
  - any — все типы файлов;
  - cell — электронные таблицы;
  - word — электронные документы;
  - images — изображения;
  - video — видеофайлы;
  - audio — аудиофайлы;
  - plugin — плагины;
  - Также можно указать конкретный тип файла, который будет возможен для выбора. Для этого перед расширением файла нужно указать символ ”□” и обернуть маску скобками. Например .getPathsByDialog(”(□.xlsx)”).
- **isMultiple** (boolean) — флаг, указывающий, можно ли выбирать несколько файлов в Проводнике.
- **isWarn** (boolean) — флаг, указывающий, будет ли макрос выдавать ошибку и останавливать выполнение кода, если файл не выбран (возвращать reject(false)). По умолчанию стоит true (для корректной работы старых макросов).

Возвращает

- **filePaths** (string[] | null) — массив строк с абсолютными путями до выбранных файлов, `null` - если установлен флаг `isWarn` как `false` и не выбраны файлы.

Пример

```
const files = await craftTechApi.getPathsByDialog('any', true);
for (let i = 0; i < files.length; i++) {
  const handler = await craftTechApi.openFile_debug(files[i]);
  console.log(`id файла с абсолютным путём ${files[i]}: ${handler}`);
  await craftTechApi.closeFile_debug();
}
```

Код выше откроет окно проводника, в котором пользователь выбирает файлы, которые хочет открыть. После выбора файлов метод вернёт массив с абсолютными путями до выбранных файлов. Методом openFile_debug() принимаем путь до каждого файла и сохраняем данные этих файлов в БД.

getSheetArray

```
await craftTechApi.getSheetArray(fileHandler, sheetName, cellRange = '');
```

Описание

Позволяет получить все строковые и числовые данные ячеек указанного диапазона на листе книги (при указании параметра cellRange, иначе вернет все данные на листе).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.
- **sheetName** (string) — название листа.
- **cellRange** (string) — не обязательный параметр, указывающий диапазон ячеек в строковом формате (например: `A1:C2`), из которых нужно получить данные.

Возвращает

- **sheetArray** (string | number | null)[][]) — двумерный массив с данными ячеек указанного листа книги.

Пример

```
const isReceived = await craftTechApi.getSheetArray(fileHandler, 'Лист1',
'A1:B3');
console.log('Данные получены: ', isReceived);
```

Ниже представлен пример вывода:

```
[
  ['Word', 'Time', null],
  ['Water', 'Day', 'Part']
]
```

Пример без указания параметра cellRange

```
const isReceived = await craftTechApi.getSheetArray(fileHandler, 'Лист1');
console.log('Данные получены: ', isReceived);
```

Ниже представлен пример вывода:

```
[
  ['Word', 'Time', null, null, 'Way'],
  ['Water', 'Day', 'Part', 'Sound', 'Work'],
  null,
  ['Place', 4444, 'Back', null, 'Thing', null, null, null, 'Name'],
  ['Sentence', 'Man', 'Line', 'Boy', 'Farm']
]
```

Массив содержит в себе данные рядов на листе «Лист 1». На примере видно, что на листе присутствуют пять рядов с данными. Если ряд пустой, то он содержит значение `null`.

Порядковый номер массива в массиве соответствует номеру ряда в документе, то есть первый массив — это данные с первого ряда, второй массив — это данные со второго ряда и так далее.

Строковые данные ячеек хранятся аналогично: первое значение в массиве соответствует значению в первой колонке, второе значение в массиве соответствует значению во второй колонке и так далее.

Из полученных данных на примере выше, мы можем сделать несколько выводов:

1. Ячейка, которая находится во втором ряду в третьей колонке содержит значение «Part»;
2. Ячейка, которая находится в четвёртом ряду в четвёртой колонке не содержит в себе никакого строкового или числового значения (но она может быть не пустая, например, она может быть окрашена в синий цвет или может быть частью таблицы);
3. Третий ряд не содержит никаких строковых или числовых данных;
4. Лист с названием «Лист 1» не содержит строковых или числовых данных ниже 5-го ряда.

getSheetsNames

```
await craftTechApi.getSheetsNames(fileHandler);
```

Описание

Возвращает список листов книги.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.

Возвращает

- **sheetNames** (string[]) — массив строк с названиями листов, либо пустой массив в случае ошибки.

Пример

```
const allSheetsNames = await craftTechApi.getSheetsNames(fileHandler);
console.log('Названия всех листов в документе: ', allSheetsNames);
```

getStyledTable

```
await craftTechApi.getStyledTable(fileHandler, sheetName);
```

Описание

Возвращает информацию о таблицах на листе

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;

Возвращает

- **tablesInfo** (object[]) — массив из объектов, в каждом из котором хранится имя таблицы под ключом `displayName` и диапазон таблицы под `ref` для каждой таблицы на листе;

Пример

```
var handler = await craftTechApi.openFile("book.xlsx")
console.log(await craftTechApi.getStyledTable(handler, 'Лист1'));
```

Вывод:

```
[
   {displayName: 'Контроль', ref: 'B4:O11'},
   {displayName: 'Таблица1', ref: 'C18:I38'}
]
```

getUsedRange

```
await craftTechApi.getUsedRange(fileHandler, sheetName)
```

Описание

Возвращает информацию о полном диапазоне ячеек, в которых есть значения во всем листе.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа.

Возвращает

- **usedRange** (object) — объект с полями:
  - range — это строка с диапазоном в формате A1:B300;
  - firstIndex — это массив длиной 2, где первый элемент — номер строки первой ячейки со значением, а второй элемент — номер столбца первой ячейки со значением;
  - lastIndex — это массив длиной 2, где первый элемент — номер строки последней ячейки со значением, а второй элемент — номер столбца последней ячейки со значением.

```
{
   range: 'A1:B300',
   firstIndex: [1,1],
   lastIndex: [29,10]
}
```

Пример

```
const fileHandler = await craftTechApi.openFile('book.xlsx');
const range = await craftTechApi.getUsedRange(fileHandler, 'Лист1');
console.log('Информация об использованных ячейках: ', range);
```

getUserName

```
await craftTechApi.getUserName();
```

Описание

Возвращает имя пользователя в системе.

Параметры

Метод не принимает параметров.

Возвращает

- **userName** (string) — имя пользователя в системе.

Пример

```
const myName = await craftTechApi.getUserName();
console.log('Моё имя: ', myName);
```

goalSeek

```
craftTechApi.goalSeek(targetCell, expectedValue, changingCell)
```

Описание

Вычисляет значения, необходимые для достижения определенной цели. Аналог метода в VBA.

Параметры

- **targetCell** (string) — адрес ячейки, значение которой мы хотим изменить до нужного (expectedValue);
- **expectedValue** (number) — целевое значение, которое должно быть достигнуто в targetCell;
- **changingCell** (string) — адрес ячейки, значение которой будет изменяться для достижения expectedValue в targetCell.

Используется для нахождения такого значения в ячейке `changingCell`, при котором значение в `targetCell` становится равным `expectedValue`.

Статичные параметры

- **Минимальное значение диапазона** - `-335543,32`
- **Максимальное значение диапазона** - `335543,32`
- **Шаг итерации** - `0,001`
- **Максимальное количество итераций** - `100`

Возвращает

- **true** или **false** — в зависимости от того, найдено решение или нет. При **true**, в `changingCell` вставляется соответствующее найденное значение. При **false**,

вставляется верхнее или нижнее максимальное значение диапазона чисел: `335543,32/-335543,32` по окончанию 100 итераций.

Важно

Ячейка по адресу, определённой, как `targetCell`, обязательно должна содержать формулу. Это распространяется так же и на оригинальный **GoalSeek** в Excel и на **GoalSeek** вызванный через интерфейс в Р7-офис

Пример

У нас есть ячейка **A1**, которая содержит формулу `=C1*2`. И мы хотим привести **A1** к **15** изменяя значения в ячейки **C1**. По итогу, число `7.499977648258209` вставляется в ячейку **C1**.

```
const res = craftTechApi.goalSeek('A1', '15', 'C1');
console.log(res); // true Решение найдено: 7.499977648258209
```

groupActive

```
craftTechApi.groupActive(sheetStr, range, coor)
```

Описание

Группирует диапазон ячеек в активном файле, из которого запущен текущий макрос.

Параметры

- **sheetStr** (string) — название листа в активном файле, из которого запущен текущий макрос;
- **range** (string) — диапазон ячеек, который необходимо сгруппировать;
- coor (‘row’ | ‘column’) — флаг, который указывает, по какой координате должна проводиться группировка: по строкам или по столбцам.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.groupActive('Лист1', 'A1:D1', 'column');
```

hideColumns

```
await craftTechApi.hideColumns(fileHandler, sheetName, cellRange, width = '');
```

Описание

Скрывает столбцы.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellRange** (string) — диапазон колонок (буквы через двоеточие в виде строкового значения);
- **width** (string) — необязательный параметр; ширина колонки.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного скрытия столбцов.

Пример

```
const isHide = await craftTechApi.hideColumns(fileHandler, 'Лист1', 'B:D');
console.log('Колонки скрылись: ', isHide);
```

hideRows

```
await craftTechApi.hideRows(fileHandler, sheetName, cellRange);
```

Описание

Скрывает строки.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellRange** (string) — диапазон строк (числа через двоеточие в виде строкового значения).

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного скрытия строк.

Пример

```
const isHide = await craftTechApi.hideRows(fileHandler, 'Лист1', '2:6');
console.log('Ряды скрыты: ', isHide);
```

infoFile

```
await craftTechApi.infoFile(filePath);
```

Описание

Используется для получения информации о файле по его абсолютному пути; также возвращаемый объект хранит текущую дату и время.

Параметры

- **dirName** (filePath) — путь до файла, по которому необходимо получить информацию.

Возвращает

- **filesInDir** (object) — объект с информацией о файле:
  - birth — дата и время создания файла;
  - size — размер файла в битах;
  - today — текущая дата и время;
  - updat — дата и время обновления (модификации) файла.

```
{
  birth: "2024-10-23T16:35:21.572021Z"
  size: 19828681
  today: "2024-10-25T10:49:44.602293Z"
  updat: "2024-10-23T16:35:21.956402Z"
}
```

Пример

```
const infoFile = await craftTechApi.infoFile('test.txt');
console.log('Информация о файле test.txt: ', infoFile);
```

inputForm

```
await craftTechApi.inputForm(obj);
```

Описание

Выводит на экран окно с полями ввода, лейблами и кнопками с возможностью кастомизации; при нажатии на выбранную кнопку запускает функцию или макрос.

Параметры

- **obj** (object) — объект в входными данными:
  - title (string) — заголовок окна;
  - input (object[]) — массив объектов с информацией о каждом поле ввода:
    - label (string) — лейбл поля ввода;
    - value (string) — дефолтное значение поля ввода.
  - actions (object[]) — массив объектов с информацией о каждой исполняемой кнопке:
    - btnCap (string) — текст кнопки, который будет виден пользователю;
    - btnVal (string) — значение, которое хранит кнопка;
    - callback (function) — функция (например, макрос), который выполняется при нажатии на эту кнопку.

Возвращает

- **result** (string[]) — массив значений, переданных пользователем в полях ввода.

Пример

Код ниже запустит окно, которое будет иметь заголовок `"Вставьте значение"`, поле ввода 1 без лейбла, но с дефолтным значением `"value1"`, поле ввода 2 с лейблом, но без дефолтного значения, а также кнопки “Принять” и “Отклонить”.

При нажатии на кнопку `"Принять"` запустится функция `doOne`, которая выведет текст `"Предложение принято!"` в консоль. При нажатии на кнопку “Отклонить” запустится функция

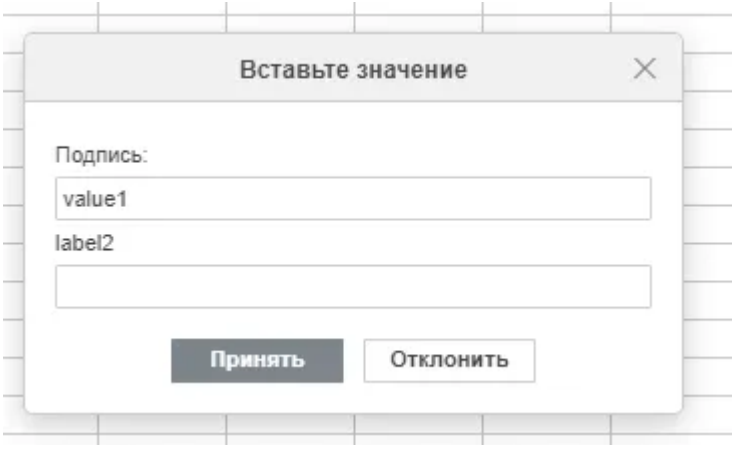
doTwo, которая выведет текст "Предложение отклонено!" в консоль. Вне зависимости от нажатой кнопки метод вернёт массив значений, переданных пользователем в полях ввода в переменную res.

```
function doOne() {
    console.log("Предложение принято!");
}

function doTwo() {
    console.log("Предложение отклонено!");
}

const obj = {
    title: "Вставьте значение",
    input: [
     { label: undefined, value: "value1" },
     { label: "label2", value: "" },
    ],
    actions: [
     { btnCap: "Принять", btnVal: "yes", callback: doOne },
     { btnCap: "Отклонить", btnVal: "no", callback: doTwo },
    ],
};

const res = await craftTechApi.inputForm(obj);
console.log(res);
```



{width="392" height="238"}

insertRowFile

```
await craftTechApi.insertRowFile(fileHandler, sheetName, rowNumber, rowCount =
1);
```

Описание

Вставляет пустую строку на указанную позицию со смещением вниз.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **rowNumber** (number) — номер строки;
- **rowCount** (number) — необязательный параметр (по умолчанию 1); количество вставленных пустых строк.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного добавления пустой строки.

Пример

```
const result = await craftTechApi.insertRowFile(fileHandler, 'Лист1', 3, 4)
console.log('Строки вставлены: ', result);
```

Код выше добавит четыре пустых ряда, начиная со строки 3, в листе «Лист1». Соответственно, все находящиеся ниже строки будут смещены на четыре позиции вниз, со строки 3 на строку 7.

isFileEncrypted

```
await craftTechApi.isFileEncrypted(filePath);
```

Описание

Проверяет, установлен ли пароль в файле.

Параметры

- **filePath** (string) — путь до файла.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае, если в файле установлен пароль, иначе `false`.

Пример

```
// const isEncrypted = await craftTechApi.encryptFile('ivan.xlsx', 'qwerty');
// console.log('Файл зашифрован: ', isEncrypted);
const result = await craftTechApi.isFileEncrypted('ivan.xlsx');
console.log('Файл защищён: ', result);
```

isFileOpen

```
await craftTechApi.isFileOpen(path);
```

Описание

Проверяет, открыт ли файл физически в данный момент времени (другим пользователем или этим же пользователем в Excel/P7-Офис).

P.S. Нужно проверять, существует ли проверяемый файл (если он создаётся динамически), например с помощью метода isFileExist(), иначе, если файла не существует, будет возвращаться true, как будто файл открыт.

Параметры

- **path** (string) — путь до файла.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае, если файл открыт, иначе `false`.

Пример

```
const isOpen = await craftTechApi.isFileOpen('ivan.xlsx');
if (isOpen) {
  console.log('Файл открыт.');
} else {
  console.log('Файл не открыт.');
}
```

isMergedCell

```
await craftTechApi.isMergedCell(fileHandler, sheetName, cellAddress)
```

Описание

Предоставляет данные об объединенных ячейках по адресу.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellAddress** (number) — адрес ячейки, которая потенциально входит в объединенный диапазон.

Возвращает

- **mergedData** (object | false) — объект, если ячейка входит в объединенный диапазон, иначе возвращает `false`. Поля возвращаемого объекта следующие:
  - isMerged — статус объединения ячеек;
  - ref — текстовый диапазон объединенных ячеек;
  - range — индексы объединенных ячеек.

Пример

```
const res = await craftTechApi.isMergedCell(fileHandler, 'Лист1', 'A1');
console.log('Ячейка объединена: ', res);
```

```
{
   isMerged: true,
   ref: 'A1:A2',
   range: '1,1:1,2'
}
```

isSheetExist

```
await craftTechApi.isSheetExist(fileHandler, sheetName);
```

Описание

Проверяет наличие листа в книге по его названию.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа.

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае наличия листа в книге, иначе `false`.

Пример

```
const fh = await craftTechApi.openFile('book.xlsx');
const check = await craftTechApi.isSheetExist(fh, 'Лист 1');
console.log('isSheetExist(): ', check);
```

isSheetProtected

```
await craftTechApi.isSheetProtected(fileHandler, sheetName);
```

Описание

Проверяет, защищён ли указанный лист переданного файла.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.
- **sheetName** (string) — название листа.

Возвращает

- **true | false** (boolean) — логическая истина `true`, если лист защищён, иначе `false`.

Пример

```
const isProtected = await craftTechApi.isSheetProtected(fileHandler, 'Отчёт');
console.log('Лист защищён: ', isProtected);
```

Если лист «Отчёт» защищён от внесения изменений, метод вернёт `true`.

macroStatus

```
await craftTechApi.macroStatus(counter, isBar, statuses);
```

Описание

Выводит на экран окно с прогресс-баром (индикатором выполнения) во время выполнения макроса.

Параметры

- **counter** (number) — счётчик, указывающий, сколько шагов выполняет макрос;
- **isBar** (boolean) — необязательный параметр; флаг, указывающий, должен ли быть визуализирован индикатор;
- **statuses** (string[]) — необязательный параметр; массив строк, каждая из которых описывает определенную стадию.

Возвращает

- **macroStatus** (macroStatusType) — экземпляр класса.

После создания экземпляра через метод `macroStatus`, можно вызывать доступные методы для управления состоянием и отображением окна с прогресс-баром.

Методы

- **nextCount()** — перейти на следующий шаг (текущий `counter` увеличится на единицу);
- **nextStage()** — перейти на следующий статус (переход на следующий элемент массива `statuses`).

Здесь важна разница между `nextCount` и `nextStage`. Есть макросы, где нет необходимо описывать текущую работу макроса столь детально. Однако ряде макросов есть такая потребность — делить макрос на стадии, внутри каждой из которых есть определённый шаги.

Например, вам необходимо перенести данные из одного файла в другой. В исходном файле четыре листа, у этих листов 15203, 29389, 390, 193 ячеек, соответственно. Тогда в `counter` передаётся значение, равное сумме количества этих ячеек, а в `statuses`, например, строки рода «Копирование %Название_листа%».

Пример

```
const craftTechApi = await getCrafttechApiAsync();
const statuses = ['Создание развёрнутой Таблицы 1', 'Создание развёрнутой Таблицы 2', 'Создание развёрнутой Таблицы 3', 'Создание развёрнутой Таблицы 4', 'Создание развёрнутой Таблицы 5'];

const macro = await craftTechApi.macroStatus(800, true, statuses);

let k = 1;
let interval = setInterval(() => {
  if (k % 1 === 0) {
    macro.nextCount();
  }

  if (k % 200 === 0) {
    macro.nextStage();
  }

  k++;

  if (k > 1000) {
      clearInterval(interval);
  }
}, 10);
```

Код выше имитирует процесс постепенного обновления прогресс-бара с переключением этапов выполнения на определённых промежутках времени. На реальных макросах применение `nextCount`/`nextStage` понадобится, например, при завершении определённой функции, цикла или другого фрагмента кода.

mathFunction

```
await craftTechApi.mathFunction(functionName, args);
```

Описание

Метод, который обращается к необходимой математической функции и возвращает результат.

Параметры

- **functionName** (string) — название функции, примеры см. Математические функции;
- **args** (аргументы) — аргументы (любое кол-во), необходимых для выполнения мат. функции. Передаются как `*args`.

Возвращает

- **value** (string) — строковое значение с посчитанным результатом. Если произошла ошибка в расчете - вернет false, в других случаях возвращает вычисленное значение в строковом представлении. Чтобы получить число **нужно распарсить** (JSON.parse) или **использовать конструктор Number**

Пример

```
const res = await crafttechApi.mathFunction("beta_inv", 0.03, 1, 5);
console.log('Данные получены: ', res);
```

Ниже представлен пример вывода:

```
Данные получены:  0.006073323851797274
```

mathFunctionMany

```
await craftTechApi.mathFunctionMany(functionName, [[args], [args], ...);
```

Описание

Метод, который обращается к необходимой математической функции и возвращает результат. Отличается от **mathFunction()** тем, что вместо простой передачи всех параметров функции в качестве 2 аргумента принимает двумерный массив, где каждый вложенный массив - аргументы для вызова функции. Количество вложенных элементов массива равняется количеству вызовов функций.

Параметры

- **functionName** (string) — название функции, примеры см. Математические функции;
- **[[args], [args], …]** (array) — двумерный массив, вложенный элемент которого содержит аргументы (любое кол-во), необходимых для выполнения мат. функции.

Возвращает

- **value** (string) — строковое значение, содержащее массив вычисленных результатов функций. Если произошла ошибка в расчете - вместо вычисляемого значения будет false, в других случаях возвращает вычисленное значение в строковом представлении. Чтобы получить число **нужно распарсить** (JSON.parse).

Пример

```
const res = await crafttechApi.mathFunctionMany("beta_inv", [[0.03, 1, 5],
[0.23, 1, 5], [0.37, 3, 7]]);
console.log('Данные получены: ', JSON.parse(res));
```

Ниже представлен пример вывода:

```
Данные получены:  [0.006073323851797274, 0.05093021979413184,
0.23953121637009903]
```

mathRand

```
await craftTechApi.mathRand(callTimes, seed);
```

Описание

Метод, который возвращает сгенерированные псевдослучайные числа. Может вызываться определенное количество раз и принимать семечко для запоминания последовательности чисел.

Параметры

- **callTimes** (number) — количество возвращаемых псевдослучайных чисел;
- **seed** (number) — семечко для запоминания последовательности генерации чисел.

Возвращает

- **value** (array) — массив сгенерированных псевдослучайных чисел.

Пример

```
const res = await crafttechApi.mathRand(5, 3);
console.log('Данные получены: ', res);
```

Ниже представлен пример вывода:

```
Данные получены:  (5) [0.08564916714362436, 0.2368105065960997,
0.8012744652063969, 0.5821620360643678, 0.09412864224039919]
```

mergeCells

```
await craftTechApi.mergeCells(fileHandler, sheetName, range)
```

Описание

Объединяет ячейки в диапазоне.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **range** (string) — диапазон ячеек, которые будут объединены.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного объединения ячеек.

Пример

```
const res = await craftTechApi.mergeCells(fileHandler, 'Лист1', 'A1:A10');
console.log('Ячейка объединена: ', res);
```

moveSheetName

```
await craftTechApi.moveSheetName(fileHandler, sheetName, newPosition);
```

Описание

Перемещает лист в книге на другую позицию.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для текущего файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **newPosition** (number) — новая позиция листа; отсчёт ведётся от 0.

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае успешного перемещения листа, иначе `false`.

Пример

```
const isMoved = await craftTechApi.moveSheetName(fileHandler, 'Лист3', 0);
console.log('Лист перемещён:', isMoved);
```

numberFormat

```
await craftTechApi.numberFormat(fileHandler, sheetName, range, flag);
```

Описание

Устанавливает формат ячейки (или диапазона ячеек) по переданному шаблону. Содержимое ячейки должно быть числом.

> Аналог **SetNumberFormat** в Р7-Офис, но работает с внешними файлами.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **range** (string) — диапазон ячеек;
- **flag (string)** — формат отображения значения в ячейке. Поддерживаются следующие флаги:
  - `'@'` — преобразование значения в текст.
  - `'0'` — округление числа до целого.
  - `'0.00'` — отображение числа с двумя знаками после запятой.
  - `'#,##0'` — отображение числа с разделением тысяч пробелами (например, 1234567 → 1 234 567).
  - `'#,##0.00'` — отображение числа с разделением тысяч пробелами и двумя знаками после запятой.
  - `'0%'` — отображение значения в виде процента без знаков после запятой.
  - `'0.00%'` — отображение значения в виде процента с двумя знаками после запятой.
  - `'0.00E+00'` — отображение значения в экспоненциальной (научной) форме.
  - `'# ?/?'` — отображение числа в виде простой дроби.
  - `'# ??/??'` — отображение числа в виде дроби с выравниванием.
  - `'mm-dd-yy'` — отображение даты в формате месяц-день-год (например, 12-31-25).

- `'d-mmm-yy'` — отображение даты с сокращённым названием месяца (например, 5-янв-25).
- `'d-mmm'` — отображение только дня и сокращённого месяца.
- `'mmm-yy'` — отображение месяца и года.
- `'h:mm AM/PM'` — отображение времени в 12-часовом формате с AM/PM.
- `'h:mm:ss AM/PM'` — отображение времени с секундами и AM/PM.
- `'h:mm'` — отображение времени в 24-часовом формате.
- `'h:mm:ss'` — отображение времени с секундами.
- `'m/d/yy h:mm'` — отображение даты и времени.
- `'#,##0 ;(#,##0)'` — отображение положительных и отрицательных чисел в формате с разделением тысяч запятыми, отрицательные числа отображаются в скобках.
- `'#,##0.00;(#,##0.00)'` — как выше, но с двумя знаками после точки.
- `'mm:ss'` — отображение времени в формате минуты:секунды.
- `'[h]:mm:ss'` — отображение продолжительности времени, включая часы свыше 24.
- `'mmss.0'` — отображение времени в виде минут и секунд с одной десятичной.
- `'##0.0E+0'` — отображение в научной нотации с одним десятичным знаком.
- `'dd/mm/yyyy;@'` — наложение маски на ячейку для корректного отображения даты в формате, используемом в Р7-Офисе.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного преобразования числа.

Пример

```
const result = await craftTechApi.numberFormat(fileHandler, 'Лист1', 'A1:A3',
'# ##0,00');
console.log('Преобразование чисел: ', result);
```

openExplorer

```
await craftTechApi.openExplorer(path);
```

Описание

Используется для нативного открытия файла (или папки) в системе. При помощи данного метода можно открывать файлы и папки с открытием окон.

Параметры

- **path** (string) — путь до файла.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного нативного открытия файла (или папки) в системе.

Пример

```
const result = await craftTechApi.openExplorer('/home/files/book1.xlsx');
console.log('Файл открыт: ', result);
```

openFile

```
await craftTechApi.openFile(path);
```

Описание

Открывает внешний файл, добавляет его в базу данных.

Параметры

- **path** (string) — путь до файла.

Возвращает

- **fileHandler** (number) — ссылка на файл; для последующих действий с файлом (изменение, получение ячеек и т.д.) для обращения к методам CraftTechPE нужно использовать ссылку на файл, а не путь до него, либо `false` в случае, если файл открыть не удалось.

Пример

```
const fh = await craftTechApi.openFile('book.xlsx');
console.log('ID файла: ', fh);
```

pasteDataFromBuffer

```
await craftTechApi.pasteDataFromBuffer(fileHandler, sheetName, cellAddress,
width = 0, height = 0);
```

Описание

Копирует данные из буфера (текст/изображение) в указанную ячейку.

> Для работы на Linux нужно установить дополнительный модуль (sudo apt-get install xclip) для работы copy/paste механизма

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellAddress** (string) — адрес ячейки (или диапазона ячеек) в строковом формате, куда нужно вставить данные;
- **width** (number) — (необязательный) ширина изображения в пикселях
- **height** (number) — (необязательный) высота изображения в пикселях

> Если не указывать width и height, то размеры изображения будут как у исходного изображения из буфера

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешной вставки данных.

Пример

```
const result = await craftTechApi.pasteDataFromBuffer(handler, 'Лист1', 'B27');
console.log('Данные вставлены: ', result);
```

protectBookActive

```
craftTechApi.protectBookActive(password)
```

Описание

Устанавливает защиту от редактирования на активный файл, из которого запущен текущий макрос.

Параметры

- **password** (string) — пароль для защиты.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.protectBookActive('crafttech123');
```

Файл станет недоступен для создания новых листов.

protectFile

Описание

protectFile(filePath, password) защищает файл от создания новых листов и получения доступа к скрытым листам.

Принимает:

- filePath (string) — путь до файла;
- password (string) — опционально; пароль для защиты.

Возвращает:

true, если операция была выполнена успешно, иначе **false**;

Пример:

```
const result = await craftTechApi.protectFile('fileToProtect.xlsx', 'qwerty')
```

Код выше защитит `fileToProtect.xlsx` файл таким образом, что пользователь не сможет просматривать скрытые листы и создавать новые. При этом у него остаётся доступ к изменению данных уже существующих видимых листов.

Если указать пароль, то при попытке снять защиту через интерфейс Р7-Офис (Защита > опция “Защитить книгу”) программа потребует пароль. Если не указывать пароль, то для снятия защиты нужно просто нажать на опцию).

protectRangeActive

```
craftTechApi.protectRangeActive(sheetStr, range, name)
```

Описание

Устанавливает защиту от редактирования на диапазон данных листа из активного файла, из которого запущен текущий макрос.

Параметры

- **sheetName** (string) — название листа;
- **range** (string) — диапазон данных, который нужно защитить;
- **name** (string) — название (идентификатор) защищённого диапазона; оно необходимо для того, чтобы мы могли снять защиту с диапазона, обратившись к его названию; у каждого диапазона название **уникальное**.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.protectRangeActive('Лист1', 'A1:A3', 'testRange');
```

Код выше защитит ячейки A1, A2, A3 от редактирования.

protectSheet

```
await craftTechApi.protectSheet(filePath, sheetName, password);
```

Описание

Устанавливает защиту на лист от редактирования.

Параметры

- **filePath** (string) — путь до файла;
- **sheetName** (string) — название листа;
- **password** (string) — пароль для защиты.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешной защиты листа. Если лист уже защищён, метод вернёт `false`, а пароль останется прежним.

Пример

```
const result = await craftTechApi.protectSheet('fileToProtect.xlsx', 'Лист1',
'qwerty');
console.log('protectSheet(): ', result);
```

Код выше защитит лист «Лист1» файла `fileToProtect.xlsx` таким образом, что пользователь не сможет изменять содержимое ячеек: при попытке сделать это выводится окно, что лист защищён и для его снятия нужен пароль.

Если указать пароль, то при попытке снять защиту через интерфейс Р7-Офис (Защита > опция “Защитить лист”) программа потребует пароль. Если не указывать пароль, то для снятия защиты нужно просто нажать на опцию).

protectSheetActive

```
craftTechApi.protectSheetActive(sheetStr, password)
```

Описание

Устанавливает защиту от редактирования на листа из активного файла, из которого запущен текущий макрос.

Параметры

- **sheetName** (string) — название листа;
- **password** (string) — пароль для защиты.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.protectSheetActive('Лист1', 'crafttech123');
```

Код выше защитит лист «Лист1» таким образом, что пользователь не сможет изменять содержимое ячеек: при попытке сделать это выводится окно, что лист защищён и для его снятия нужен пароль.

recalcAllFormulas

```
await craftTechApi.recalcAllFormulas(filePath);
```

Описание

Пересчитывает все формулы во внешнем документе.

Параметры

- **filePath** (string) — путь до файла.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного пересчёта формул в файле, иначе `false`.

Пример

```
const result = await craftTechApi.recalcAllFormulas('crafttech.xlsx');
console.log('Пересчёт формул прошёл успешно: ', result);
```

removeAutoFilter

```
craftTechApi.removeAutoFilter(handler, sheetName)
```

Описание

Сбрасывает фильтрацию на листе.

> Чтобы показать скрытые фильтром строки стоит использовать showRows;

> Для активного файла можно использовать resetAutofilterActive;

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа на котором нужно сбросить все автофильтры;

Возвращает

true | false (boolean) — логическая истина `true` в случае успешного удаления листа, иначе `false`.

Пример

```
craftTechApi.removeAutoFilter(handler, 'Лист1');
```

renameFile

```
await craftTechApi.renameFile(oldName, newName);
```

Описание

Переименовывает файл.

Параметры

- **oldName** (string) — путь до файла со старым именем;
- **newName** (string) — путь до файла с новым именем.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного переименования файла, иначе `false`.

Пример

```
const isRenamed = await craftTechApi.renameFile('files/test.js',
'files/testNew.txt');
console.log('renameFile():', isRenamed);
```

renameSheet

```
await craftTechApi.renameSheet(fileHandler, oldName, newName);
```

Описание

Переименовывает лист в книге.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для текущего файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **oldName** (string) — прежнее название листа;
- **newName** (string) — новое название листа.

Возвращает

- **true | false** (boolean) — возвращает логическую истину `true` в случае успешного переименования листа, иначе `false`.

Пример

```
const fh = await craftTechApi.openFile('book.xlsx');
await craftTechApi.renameSheet(fh, 'Лист 1', 'Total');
await craftTechApi.saveFile(fh);
```

reRenderSheet

```
craftTechApi.reRenderSheet();
```

Описание

Обновляет данные на текущем листе файла, из которого запущен текущий макрос.

Параметры

Метод не принимает параметров.

Возвращает

Метод ничего не возвращает.

Пример

```
const sheet1 = Api.GetSheet('Лист1');
Api.AddSheet('Лист2');
const sheet2 = Api.GetSheet('Лист2');
sheet2.SetActive();
sheet1.SetName('test');
```

В этом коде мы создаём новый лист «Лист2», а лист «Лист1» переименовываем в «test». После выполнения кода кажется, что «Лист1» остался с прежним названием, но если создадим новый лист вручную (через +), либо запустим макрос ещё раз, то название листа уже поменялось. Эти изменения вписались, но не сразу изменились.

Дело в том, что в Р7-Офис повторный рендеринг текущего состояния листа зачастую происходит лишь после проделывания каких-то действий вручную. Для нас это может быть критично, поэтому был создан метод, обновляющий данные на текущем листе.

```
const sheet1 = Api.GetSheet('Лист1');
Api.AddSheet('Лист2');
const sheet2 = Api.GetSheet('Лист2');
sheet2.SetActive();
sheet1.SetName('test');

craftTechApi.reRenderSheet();
```

Теперь лист переименуется сразу.

resetAutofilterActive

```
craftTechApi.resetAutofilterActive(sheetStr)
```

Описание

Сбрасывает фильтрацию в активном файле, из которого запущен текущий макрос.

Параметры

- **sheetStr** (string) — название листа в активном файле, из которого запущен текущий макрос.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.resetAutofilterActive('Лист1');
```

resetFilters

```
await craftTechApi.resetFilters(fileHandler, sheetName)
```

Описание

Сбрасывает фильтрацию с указанного листа.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного сброса фильтрации.

Пример

```
const fh = await craftTechApi.openFile('book.xlsx');
const check = await craftTechApi.resetFilters(fh, 'Лист1');
console.log('resetFilters(): ' + check);
```

resizeActiveFileStyledTable

```
await craftTechApi.resizeActiveFileStyledTable(sheetName, cell, newRange);
```

Описание

Изменяет размер стилизованной таблицы в активном файле.

> Метод реализован с использованием манипуляций с пользовательским меню. Внимательно проверяйте входные данные!

Параметры

- **sheetName** (string) — Имя листа.
- **cell** (string) — Ячейка находящаяся в границах стилизованной таблицы, границы которой требуется изменить.
- **newRange** (string) — Новый диапазон стилизованной таблицы.

Возвращает

- **true / false** (boolean) — возвращает логическую истину в случае успешного выполнения метода.

Пример

```
  const sheetsArr = ['Лист1', 'Лист2', 'Лист3'];
  for (const sheet of sheetsArr) {
    const cell = 'C1';
    const newRange = 'C1:G16';
    const isStyledTableResized = await craftTechApi.resizeActiveFileStyledTable(
      sheet,
      cell,
      newRange
    );
    console.log(
      `На листе ${sheet} стилизованная таблица на позиции ${cell} обновлена:
${isStyledTableResized}`
    );
  }
```

runFile

```
await craftTechApi.runFile(path);
```

Описание

Запускает исполняемую программу.

Параметры

- **path** (string) — путь до файла.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного запуска исполняемой программы.

Пример

```
const result = await craftTechApi.runFile('/home/files/testFile.exe');
console.log('Программа запущена: ', result);
```

saveActiveFile

```
await craftTechApi.saveActiveFile()
```

Описание

Принудительно сохраняет активный файл, из которого запущен текущий макрос.

Параметры

Метод не принимает параметров.

Возвращает

Метод ничего не возвращает.

Пример

```
await craftTechApi.saveActiveFile();
```

saveFile

```
await craftTechApi.saveFile(fileHandler);
```

Описание

Сохраняет файл (и изменения в нём) в базу данных.

> Убедитесь, что файл, в который Вы внесли изменения и хотите сохранить, нигде не открыт явно (в Р7-Офис, Microsoft Office или Блокноте), иначе изменения не будут внесены.

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного сохранения файла, иначе `false`.

Пример

```
const fh = await craftTechApi.openFile('book.xlsx');
await craftTechApi.editCellValue(fh, 'Лист1', 'A1', '3');
await craftTechApi.saveFile(fh);
```

selectRangeDialog

```
await craftTechApi.selectRangeDialog(title = "Введите диапазон данных");
```

Описание

Выводит на экран окно с полем ввода, которое содержит диапазон данных, выбранный пользователем.

Параметры

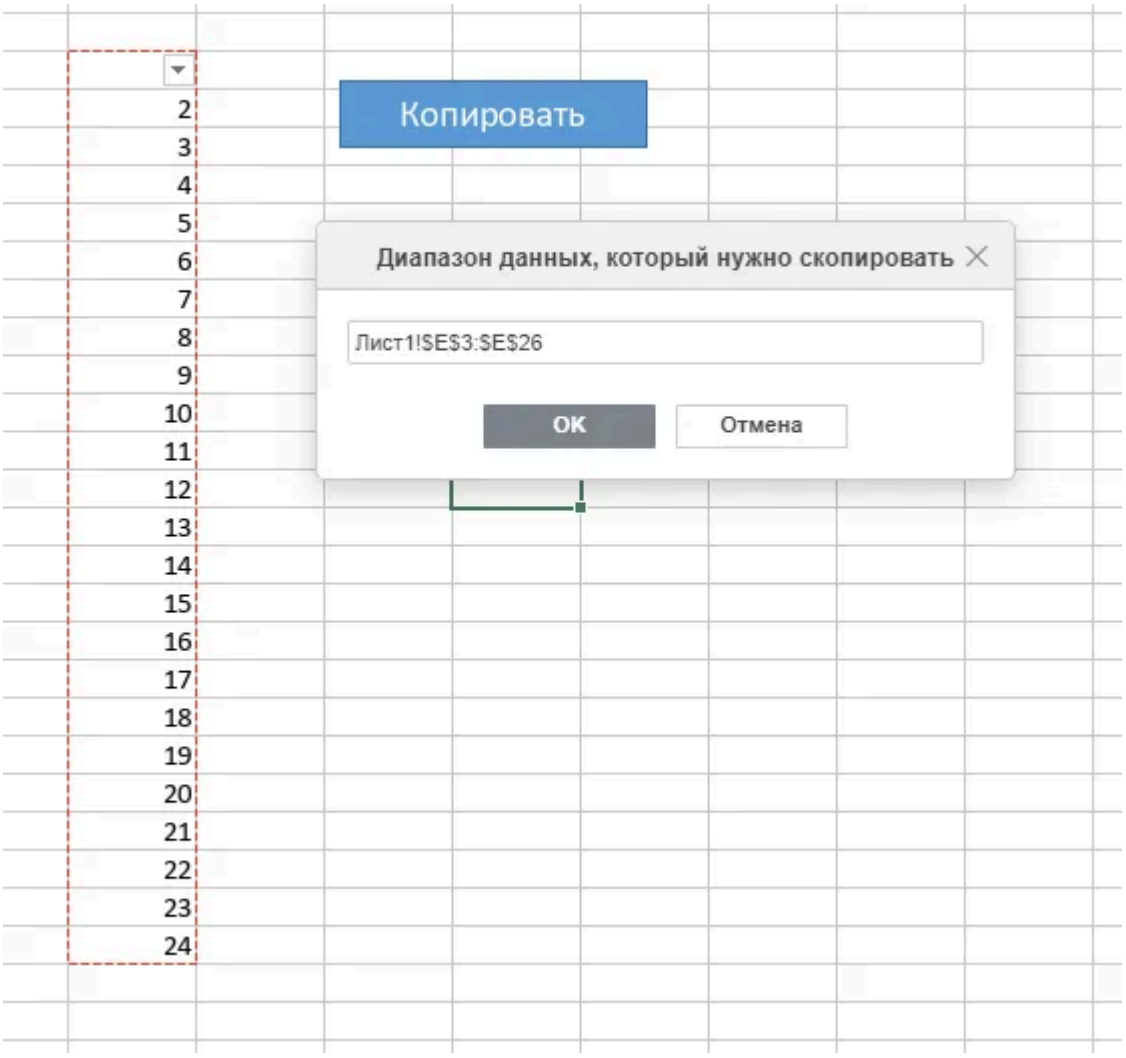
- **title** (string) — необязательный параметр; заголовок окна; чаще всего здесь указывают название макроса.

Возвращает

- **result** (string) — значение, переданное пользователем в поле ввода.

Пример

```
const data = await craftTechApi.selectRangeDialog('Диапазон данных, который
нужно скопировать');
console.log(data);
```



{width=420 height=393}

sendByMail

```
await craftTechApi.sendByMail(title, body, email, fileName = 'null');
```

Описание

Отправляет письмо с вложенными файлами на электронную почту через вызов Thunderbird.

> Поддерживается только на Astra Linux.

> Не путать с sendMail, который использует ручную настройку почтового сервиса через SMTP-сервер для отправки писем.

Параметры

- **title** (string) — заголовок письма;
- **body** (object) — тело письма, объект с полями:
  - text — текст, **оставляем пустой строкой**;
  - html — HTML-разметка;
- **emails** (string) — массив с максимум тремя элементами:
  - recipients — список получателей (обязательный), перечисляются в массиве строк, строки без пробелов;
  - cc — список получателей в копии (либо пустой массив), перечисляются в массиве строк, строки без пробелов;
  - bcc — скрытые скопии (либо пустой массив), перечисляются в массиве строк, строки без пробелов;
- **fileName** (string) — пути до файлов **одной строкой через запятую**, которые будут приложены к письму. Если вложений нет - передаётся пустая строка.

Пример тела письма:

```
const body = {
  text: '',
  html: `
  <h1>Здравствуйте</h1>
  <p>
    Я ваше <strong>письмо</strong>
  </p>
  `,
}
```

Пример массива с электронными почтами:

```
[
  ['mainmail@mail.com', 'mainmailtwo@mail.com'], // получатели
  ['copymail@mail.com', 'copymailtwo@mail.com'], // копии
  ['blindcopymail@mail.com', 'blindcopymailtwo@mail.com'], // скрытые копии
]
```

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного отправления письма.

Пример

```
const result = await craftTechApi.sendByMail(
  'Заголовок письма',
  body,
  [
     ['mainmail@mail.com', 'mainmailtwo@mail.com'],
     ['copymail@mail.com', 'copymailtwo@mail.com'],
     ['blindcopymail@mail.com', 'blindcopymailtwo@mail.com'],
  ],
  ['file.md', 'file.py'].join(','),
)

if (result) {
  console.log('Письмо успешно отправлено!')
} else {
  console.log('Письмо не отправлено.')
}
```

Пример 2 (только основной адресат)

```
const result = await craftTechApi.sendByMail(
  'Заголовок письма',
{
  text: '',
  html: `<h1>Здравствуйте</h1>`,
}
[
     ['mainmail@mail.com', 'mainmailtwo@mail.com'],
     [],
     [],
  ],
  ['file.md', 'file.py'].join(','),
)
```

Пример 3 (только основной адресат)

```
const result = await craftTechApi.sendByMail(
  'Заголовок письма',
{
  text: '',
  html: `<h1>Здравствуйте</h1>`,
}
[
     ['mainmail@mail.com', 'mainmailtwo@mail.com']
  ],
  ['file.md', 'file.py'].join(','),
)
```

sendMail

```
await craftTechApi.sendMail(title, body, emails, fileName, conf);
```

Описание

Отправляет письмо с вложенными файлами на электронную почту через SMTP-сервер.

> Не путать с sendByMail, который использует Р7-Органайзер для отправки писем.

Параметры

- **title** (string) — заголовок письма;
- **body** (object) — тело письма, объект с полями:
  - text — текст, **оставляем пустой строкой**;
  - html — HTML-разметка;
- **emails** (string) — массив с тремя строками; каждая такая строка написана через запятую без пробелов:
  - recipients — список получателей;
  - cc — список получателей в копии;
  - bcc — скрытые скопии;
- **fileName** (string) — пути до файлов **одной строкой через запятую**, которые будут приложены к письму;
- **conf** (obj) — настройки для почтового сервиса, с которого будет отправляться письмо:
  - host — сервер исходящей почты (SMTP-сервер);
  - port — порт;
  - auth — аутентификация:
    - user — имя пользователя;
    - pass — пароль **приложения**.

Пример тела письма:

```
const body = {
  text: '',
  html: `
  <h1>Здравствуйте</h1>
  <p>
    Я ваше <strong>письмо</strong>
  `,
}
```

Пример массива с электронными почтами:

```
[
  ['mainmail@mail.com'].join(','), // получатели
  ['copymail@mail.com', 'copymailtwo@mail.com'].join(','), // копии
  ['blindcopymail@mail.com', 'blindcopymailtwo@mail.com'].join(','), // скрытые
копии
]
```

Пример настройки для почтового сервиса:

```
const conf = {
  host: 'smtp.mail.ru',
  port: 465,
  auth: {
    user: 'r7alerts@mail.ru',
    pass: 'SYE0bc7MYsc0sJTP1Bec'
  },
}
```

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного отправления письма.

Пример

```
const result = await craftTechApi.sendMail(
  'Заголовок письма',
  body,
  [
     ['mainmail@mail.com'].join(','),
     ['copymail@mail.com', 'copymailtwo@mail.com'].join(','),
     ['blindcopymail@mail.com', 'blindcopymailtwo@mail.com'].join(','),
  ],
  conf,
)

if (result) {
  console.log('Письмо успешно отправлено!')
} else {
  console.log('Письмо не отправлено.')
}
```

setAutofilterActive

```
craftTechApi.setAutofilterActive(sheetStr, range)
```

Описание

Устанавливает фильтрацию в активном файле, из которого запущен текущий макрос.

Параметры

- **sheetStr** (string) — название листа в активном файле, из которого запущен текущий макрос;
- **range** (string) — диапазон ячеек, которому необходимо установить автофильтр.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.setAutofilterActive('Лист1', 'A1:D15');
```

setAutoFitCell

```
await craftTechApi.setAutoFitCell(filePath, sheetName, cellAddress, isRow,
isColumn);
```

Описание

Изменяет ширину столбцов или высоту строк в диапазоне под размер содержимого ячеек.

Параметры

- **filePath** (string) — путь до файла;
- **sheetName** (string) — название листа;
- **cellAddress** (string) — адрес ячейки;
- **isRow** (boolean) — указывает, будет ли высота строк подбираться автоматически;
- **isColumn** (boolean) — указывает, будет ли ширина столбцов подбираться автоматически;

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешной работы метода, иначе `false`.

Пример

```
const result = await craftTechApi.setAutoFitCell('crafttech.xlsx', 'Лист1',
'A1', 0, 1);
console.log('setAutoFitCell(): ', result);
```

setCellSize

```
await craftTechApi.setCellSize(fileHandler, sheetName, cellAddress, sizes);
```

Описание

Позволяет изменить высоту и ширину ячейки (т.е. ее строки и колонки соответственно). Размеры принимаются в пунктах для высоты и в символах для ширины. (Такие величины используются в MS Excel и Р7-Офис соответственно)

> Совет: если вам необходимо изменить размеры у диапазона ячеек (например, возьмем диапазон A3:D20), то вы можете изменить размеры ячеек только с боковой стороны (A3:A20 и A3:D3), остальные ячейки будут иметь идентичный размер.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (fileID) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — имя листа, на котором необходимо изменить размер ячейки;
- **cellAddress** (string) — координаты или адрес ячейки, размеры которой необходимо изменить;
- **sizes** (object) — объект, который имеет два поля:
  - **height** (number) — высота ячейки в пунктах;
  - **width** (number) — ширина ячейки в символах.

> Дополнение: объект **sizes** может включать в себя как ширину и высоту одновременно, так и один из параметров. Пустой объект будет просто пропущен, метод вернет `true`.

Возвращает

- **true / false** (boolean) — возвращает логическую истину в случае успешного выполнения метода.

Пример

```
let sizes = {
  height: 35.75,
  width: 12.45
}
const result = await craftTechApi.setCellSize(fileHandler, 'Лист1', 'G5', sizes);
console.log(result);
```

showColumns

```
await craftTechApi.showColumns(fileHandler, sheetName, cellRange = 'null',
width = 'null');
```

Описание

Раскрывает скрытые столбцы.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellRange** (string) — необязательный параметр; диапазон колонок (буквы, либо числа через двоеточие в виде строкового значения); если параметр не передаётся, то раскрываются все скрытые столбцы на листе;
- **width** (string) — необязательный параметр; ширина колонки.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного отображения скрытых столбцов.

Пример

```
const isShow = await craftTechApi.showColumns(fileHandler, 'Лист1', 'A:C', 10);
// либо 1:3 альтернативно A:C
console.log('Колонки раскрылись: ', isShow);
```

showInputBox

```
await craftTechApi.showInputBox(label, title, error, value);
```

Описание

Выводит на экран окно с полем ввода для указания в нём данных пользователем.

Параметры

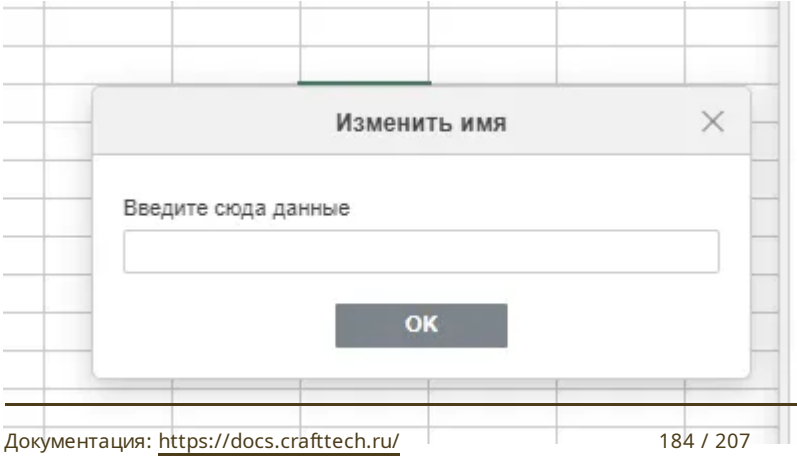
- **label** (string) — подпись к полю ввода;
- **title** (string) — заголовок окна; чаще всего здесь указывают название макроса.
- **error** (string) — необязательный параметр; сообщение, которое выводится пользователю при некорректном указании данных;
- **value** (string) — необязательный параметр; предустановленное значение поля ввода.

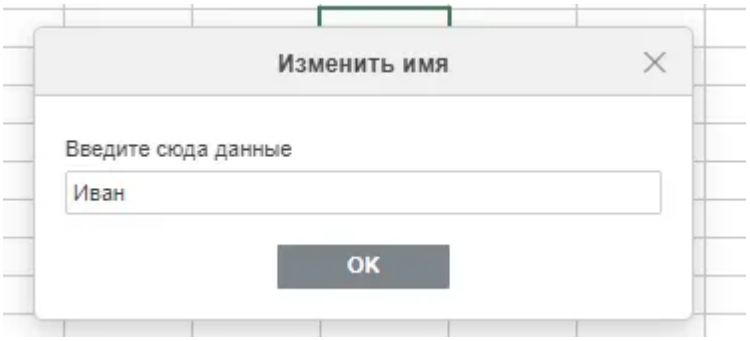
Возвращает

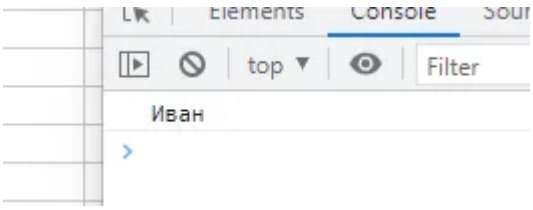
- **result** (string) — значение, переданное пользователем в поле ввода.

Пример

```
const data = await craftTechApi.showInputBox('Введите сюда данные', 'Изменить имя');
console.log(data);
```




Изменить имя
Введите сюда данные
Иван
OK


Elements
Console
top
Filter
Иван

showLevels

```
await craftTechApi.showLevels(fileHandler, sheetName, rowLevel, colLevel)
```

Описание

Отображает указанное количество уровней строк и (или) столбцов структуры.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **rowLevel** (number) — указывает количество уровней строк структуры для отображения; если этот аргумент равен 0, то произойдет скрытие всех группировок;
- **colLevel** (number) — указывает количество уровней столбцов структуры для отображения; если этот аргумент равен 0, то произойдет скрытие всех группировок.

Возвращает

- **true | false** (boolean) — логическая истина в случае успешного отображения уровней.

Пример

```
const res = await craftTechApi.showLevels(fileHandler, 'Лист1', 0, 2);
console.log('Отображение уровней произошло успешно: ', res);
```

Метод `showLevels` отображает указанное количество уровней строк и (или) столбцов структуры.

showRows

```
await craftTechApi.showRows(fileHandler, sheetName, cellRange = 'null');
```

Описание

Раскрывает скрытые строки.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **cellRange** (string) — необязательный параметр; диапазон строк (числа через двоеточие в виде строкового значения); если параметр не передаётся, то раскрываются все скрытые строки на листе.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного отображения скрытых строк.

Пример

```
const isShow = await craftTechApi.showRows(fileHandler, 'Лист1', '3:7');
console.log('Ряды раскрылись: ', isShow);
```

sortActiveSheet

```
craftTechApi.sortActiveSheet(sheetStr, start, order, expand = true)
```

Описание

Устанавливает сортировку в активном файле, из которого запущен текущий макрос.

Параметры

- **sheetStr** (string) — название листа в активном файле, из которого запущен текущий макрос;
- start (string) — адрес ячейки, откуда начинается сортировка;
- order (string) — порядок сортировки;
  - `asc` — в порядке возрастания;
  - `desc` — в порядке убывания;
  - `fill` — по цвету фона ячейки;
  - `font` — по цвету текста ячейки.
- expand (boolean) — флаг, который определяет, каким образом должна меняться сортировка в таблице относительно отсортированного столбца:
  - `true` (по умолчанию), если меняется порядок строк в таблице относительно сортировки выбранного столбца;
  - `false`, если должна произвестись сортировать только выбранного столбца, а все остальные столбцы остаются на прежнем месте.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.sortActiveSheet('Лист1', 'D1', 'asc', true);
```

spinner

```
await craftTechApi.spinner(array)
```

Описание

Отображает окно с индикатором выполнения (спиннером) и может показывать сообщения об успехе или ошибке в зависимости от условий. Метод асинхронный и резолвится только после того, как сообщение будет отображено.

Параметры

- **array** (object[]) — необязательный параметр; массив из трёх объектов с параметрами заголовка и сообщения для разных этапов (спиннер, успех, ошибка). Каждый объект имеет поля title и msg:

```
{
  title: string; // Заголовок сообщения
  msg: string;   // Текст сообщения
};
```

Первый объект — заголовок и текст, отображаемые в сообщении со спиннером. Второй объект — заголовок и текст, отображаемые в сообщении о завершении. Третий объект — заголовок и текст, отображаемые в сообщении об ошибке.

Если массив не передан или не переданы объекты (или какое-то из их свойств) - будут назначены дефолтные значения:

```
[
  { title: 'R7 Office', msg: 'Выполнение макроса, не закрывайте окно...' },
  { title: 'R7 Office', msg: 'Готово!' },
  { title: 'R7 Office', msg: 'Ошибка выполнения!' }
];
```

Возвращает

- Экземпляр **SimpleSpinner**, в котором доступны методы `done()`, `successMsg()`, `errorMsg()` и `newStatus()`.

Методы

- **done()** — закрывает окно спиннера;
- **successMsg()** — закрывает окно спиннера и показывает сообщение об успешном завершении.
- **errorMsg(err?: string)** — закрывает окно спиннера и показывает сообщение об ошибке. Если параметр не передан, используется значение из `array` либо дефолтное;
- **newStatus(newMsg: string)** — обновляет текст внутри спиннера;

Пример № 1

```
let spinner = await craftTechApi.spinner();
setTimeout(() => spinner.done(), 2000);
```

Пример № 2

```
let value = null;
let spinner = await craftTechApi.spinner(
  [
    { title: 'Заголовок спиннера', msg: 'Сообщение спиннера' },
    { title: 'Заголовок сообщения об успехе', msg: 'Сообщение об успехе' },
    { title: 'Заголовок сообщения об ошибке', msg: 'Сообщение об ошибке' },
  ],
  () => value === 1,
  true
);
setTimeout(() => (value = 1), 2000);
```

Пример № 3

```
let spinner = await craftTechApi.spinner();

(async () => {
  try {
    await new Promise((_, rj) =>
      setTimeout(() => rj(new Error('Критическая ошибка!')), 2000)
    );
  } catch (error) {
    spinner.errorMsg(error.message);
  }
})();
```

unfreezeField

```
await craftTechApi.unfreezeField(fileHandler, sheetName, field = 'null');
```

Описание

Открепляет липкие строки и столбцы.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **field** (string) — флаг для уточнения координаты:
  - all — и по строке, и по столбцу;
  - row — по строке;
  - column — по столбцу.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного открепления ячеек.

Пример

```
const freeze = await craftTechApi.unfreezeField(fileHandler, 'Лист1', 'row');
console.log('Ряд откреплён: ', freeze);
```

ungroupActive

```
craftTechApi.ungroupActive(sheetStr, range, coor)
```

Описание

Разгруппировывает диапазон ячеек в активном файле, из которого запущен текущий макрос.

Параметры

- **sheetStr** (string) — название листа в активном файле, из которого запущен текущий макрос;
- **range** (string) — диапазон ячеек, который необходимо разгруппировать;
- **coor** (‘row’ | ‘column’) — флаг, который указывает, по какой координате должна проводиться разгруппировка: по строкам или по столбцам.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.ungroupActive('Лист1', 'A1:D1', 'column');
```

unmergeCells

```
await craftTechApi.unmergeCells(fileHandler, sheetName, range)
```

Описание

Разъединяет ячейки в диапазоне.

> Требует сохранения при помощи saveFile (для внешнего файла), либо applyActiveFileChanges (для активного файла).

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **range** (string) — диапазон ячеек, которые будут разъединены.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного разъединения ячеек.

Пример

```
const res = await craftTechApi.unmergeCells(fileHandler, 'Лист1', 'A1:A10');
console.log('Ячейка разъединена: ', res);
```

unProtectFile

Описание

unProtectFile(filePath, password) снимает защиту с файла.

Принимает:

- filePath (string) — путь до файла;
- password (string) — пароль для снятия защиты (не указывать, если для защиты не задавался пароль).

Возвращает:

true, если операция была выполнена успешно, иначе **false**;

Пример:

```
const result = await craftTechApi.unProtectFile('fileToProtect.xlsx', 'qwerty')
```

Код выше снимет защиту с файла `fileToProtect.xlsx`. Добавление новых листов и просмотр скрытых листов вновь станет доступным.

unprotectRangeActive

```
craftTechApi.unprotectRangeActive(name)
```

Описание

Снимает защиту от редактирования на диапазон данных листа из активного файла, из которого запущен текущий макрос.

Параметры

- **name** (string) — название (идентификатор) защищённого диапазона; оно необходимо для того, чтобы мы могли снять защиту с диапазона, обратившись к его названию; у каждого диапазона название **уникальное**.

Возвращает

Метод ничего не возвращает.

Пример

```
craftTechApi.unprotectRangeActive('testRange');
```

Код выше снимает защиту с ячеек A1, A2, A3 от редактирования (при условии, если для защиты мы им присваивали имя `testRange`).

unProtectSheet

```
await craftTechApi.unProtectSheet(filePath, sheetName, password);
```

Описание

Снимает защиту с листа от редактирования.

Параметры

- **filePath** (string) — путь до файла;
- **sheetName** (string) — название листа;
- **password** (string) — пароль для снятия защиты (не указывать, если для защиты не задавался пароль).

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного снятия защиты с листа. Если переданный третьим параметром пароль не подходит, метод вернёт `false`, а лист останется защищённым.

Пример

```
const result = await craftTechApi.unProtectSheet('fileToProtect.xlsx', 'Лист1',
'qwerty');
console.log('unProtectSheet(): ', result);
```

Код выше снимет защиту с листа ‘Лист1’ файла `fileToProtect.xlsx`. Изменение листа ‘Лист1’ вновь станет доступным.

updateActiveFilePivotTable

```
await craftTechApi.updateActiveFilePivotTable(sheetName);
```

Описание

Обновляет сводную таблицу в активном файле.

> Требуется, чтобы лист НЕ был скрытым.

Параметры

- **sheetName** (string) — название листа.
- **cell** (string) — ячейка, которая находится в границах сводной таблицы.

Возвращает

- **true | false** (boolean) — логическая истина `true` в случае успешного обновления сводной таблицы

Пример

```
  const sheetsArr = ['Лист1', 'Лист2', 'Лист3'];
  for (const sheet of sheetsArr) {
    const cell = 'C11';
    const isDrawingsDeleted = await craftTechApi.updateActiveFilePivotTable(sheet,
cell);
    console.log(
      `На листе ${sheet} сводная таблица на позиции ${cell} обновлена:
${isDrawingsDeleted}`
    );
  }
```

updateStyledTable

```
await craftTechApi.updateStyledTable(fileHandler, sheetName, config);
```

Описание

Обновляет данные стилизованной таблицы

Параметры

- **fileHandler** (number) — ссылка на файл; ссылку предоставляет метод openFile;
- **sheetName** (string) — название листа;
- **config** (object) (Необязательный параметр) — объект, который имеет поля:
  - **displayName** (string) — название стилизованной таблицы которую необходимо изменить;
  - **ref** (string) — диапазон ячеек (например: `A1:C2`);

> Если нет config, то границы обновляются по использованному диапазону, как в getUsedRange;

> **Если больше 1й таблице на листе, то всегда используйте config**

> В дальнейшем config будет содержать различные параметры таблицы

Возвращает

- **true / false** (boolean) — возвращает логическую истину в случае успешного выполнения метода.

Пример

```
var handler = await craftTechApi.openFile("book.xlsx")
let config = {
  ref: 'B18:D25',
  displayName: "Таблица1"
}
await craftTechApi.updateStyledTable(handler, 'Лист1', config);
```

XML

Описание

В данной разметке представлены асинхронные методы CraftTechAPI для формирования контента XML-файла. Они возвращают классы, экземпляры которых позволяют создавать разметку для XML-файла.

class XML

```
const XML = await craftTechApi.XMLClass();
```

Класс XML — это класс, экземпляр которого служит родительским для формирования содержимого XML-файлов.

Методы

XML

```
let xml = new XML();
```

Конструктор класса, который используется для инициализации экземпляра.

- Принимает:
  - **rootTag** (тип XMLTag) — необязательный параметр; корневой тэг файла;
- Возвращает:
  - **экземпляр** класса XML.

setRoot

```
xml.setRoot(rootTag);
```

Сеттер, который позволяет установить корневой тэг файла.

- Параметры:
  - **root** (тип XMLTag) — корневой тэг файла;
- Возвращает:

- Метод ничего не возвращает.

removeRoot

```
xml.removeRoot();
```

Очищает корень файла. Применяется, если необходимо очистить содержимое.

- Параметры:
  - Метод не принимает параметры.
- Возвращает:
  - Метод ничего не возвращает.

getRoot

```
xml.getRoot();
```

Используется для получения корня файла.

- Параметры:
  - Метод не принимает параметры.
- Возвращает:
  - Метод возвращает корень файла (экземпляр класса XMLTag).

getContent

```
let content = xml.getContent();
console.log(content);
```

Возвращает текстовое содержимое XML-файла, сформированное согласно root tag и его дочерним объектам.

- Параметры:
  - Метод не принимает параметры.
- Возвращает:
  - **textData** (string) — текстовое содержимое XML-файла.

class XMLTag

```
const XMLTag = await craftTechApi.XMLTagClass();
```

Класс XMLTag — это *дочерний* класс, который позволяет создавать вложенность элементов в XML-файле.

Методы

XMLTag

```
let rootTag = new XMLTag("rootTag");
```

Конструктор класса, который используется для инициализации экземпляра.

- Принимает:
  - **tag** (string) — название тэга;
  - **attrs** (object) — необязательный параметр; хранятся атрибуты тэга по ключу и значению;
  - **children** (array) — необязательный параметр; массив экземпляров XMLTag, которые будут являться его дочерними элементами;
  - **content** — необязательный параметр; содержимое тэга.

    > Обратите внимание, что при формировании содержимого XML-файла поле **content** имеет приоритет перед полем **children**, что означает полное игнорирование дочерних элементов в пользу содержимого тэга.

- Возвращает:
  - **экземпляр** класса XMLTag.

addChild

```
let xmlTag = new XMLTag("tag", { attr1: "val1" }, [], "content");
rootTag.addChild(xmlTag);
```

Позволяет добавить дочерний элемент в тэг.

- Параметры:
  - **child** (тип XMLTag) — дочерний элемент, который нужно добавить в тэг.
- Возвращает:
  - Метод ничего не возвращает.

removeChild

```
rootTag.removeChild(xmlTag);
```

Удаляет дочерний элемент из массива дочерних элементов, если он входит в их число.

- Параметры:
  - **child** (тип XMLTag) — дочерний элемент, который нужно удалить.
- Возвращает:
  - Метод ничего не возвращает.

getChildren

```
rootTag.getChildren();
```

Позволяет получить все дочерние тэги для конкретного экземпляра класса.

- Параметры:
  - Метод не принимает параметры.
- Возвращает:
  - Метод возвращает массив экземпляров класса XMLTag, которые являются дочерними элементами тэга.

addAttr

```
rootTag.addAttr("attribute", "attrValue");
```

Добавляет атрибут для тэга.

- Параметры:
  - **name** - строковое название атрибута;

    - **value** - значение атрибута**.**

- Возвращает:
- Метод ничего не возвращает.

deleteAttr

```
rootTag.deleteAttr("attribute");
```

Удаляет атрибут тэга по его названию.

- Параметры:
    - **name** - строковое название атрибута;
- Возвращает:
    - Метод ничего не возвращает.

setAttrs

```
let attrs = {
  attr1: "value1",
  attr2: "value2",
  attr3: "value3",
};
rootTag.setAttrs(attrs);
```

Заменяет атрибуты тэга на переданные.

- Параметры:
    - **attrs** (object) — объект, который хранит в себе ключ-значение атрибута.
- Возвращает:
    - Метод ничего не возвращает.

setContent

```
rootTag.setContent("tagContent");
```

Устанавливает содержимое тэга.

- Параметры:

    - **content** (string | number) — содержимое, которое хотим добавить в тэг.
- Возвращает:
    - Метод ничего не возвращает.

removeContent

```
rootTag.removeContent();
```

Удаляет содержимое тэга.

- Параметры:
    - Метод не принимает параметры.
- Возвращает:
    - Метод ничего не возвращает.

getContent

```
let content = rootTag.getContent();
console.log(content);
```

Получает содержимое тэга и его дочерних тэгов в формате XML.

- Параметры:
    - Метод не принимает параметры.
- Возвращает:
    - Содержимое тэга и его дочерних тэгов в формате XML.

Пример

```
const XML = await craftTechApi.XMLClass();
const XMLTag = await craftTechApi.XMLTagClass();
let xml = new XML();
let rootTag = new XMLTag("rootTag");
xml.setRoot(rootTag);
let tagsObject = {
  tag1: {
    attr1: "stroka",
    attr2: 2,
    attr3: "new",
  },
  nexml: {
    attr1: "stroka2",
    attr2: 3,
    attr3: "new2",
  },
};
for (const [key, value] of Object.entries(tagsObject)) {
  let xmlTag = new XMLTag(key, value);
  rootTag.addChild(xmlTag);
  let xmlChild = new XMLTag(`${key}Child`, value);
  xmlTag.addChild(xmlChild);
  let xmlChild2 = new XMLTag(`${key}ChildAttr`, value, [], "myContent");
  xmlTag.addChild(xmlChild2);
}
let content = xml.getContent();
console.log(content);
```