



JS FileStorage API

File Storage API — интерфейс для работы с файловой системой. Представляет набор функций для выполнения различных операций с файлами и папками непосредственно из узла JavaScript.

Для использования File Storage API необходимо импортировать встроенный модуль `builtIn/FS` :

```
import * as fs from "builtIn/FS";
```

Описание объектов

Stats

▼ Описание

```
class Stats {  
  isFile(): boolean; // Проверка, объект - файл?  
  isDirectory(): boolean; // Проверка, объект - папка?  
  isSymbolicLink(): boolean; // Проверка, объект - символическая  
  ссылка?  
  
  mode: number; // Битовое поле, описывающее тип файла и режим  
  доступа  
  size: number; // Размер файла в байтах  
  atime: Date; // Время последнего доступа к объекту  
  mtime: Date; // Время последней модификации объекта  
  ctime: Date; // Время последнего изменения объекта  
  birthtime: Date; // Время создания объекта  
}
```

Объект Stats предоставляет информацию об объекте файловой системы.

FileHandle

▼ Описание

```
class FileHandle {  
    valueOf(): number; // Значение файлового дескриптора  
}
```

Объект FileHandle — представляет файловый дескриптор.

Dirent

▼ Описание

```
class Dirent {  
    isFile(): boolean; // Проверка, является ли объект обычным файлом  
    isDirectory(): boolean; // Проверка, является ли объект папкой  
    isSymbolicLink(): boolean; // Проверка, является ли объект  
    символической ссылкой  
    name: string; // Имя объекта  
}
```

Объект Dirent — представление элемента каталога, который может быть файлом или подкаталогом внутри каталога.

Функции File Storage

▼ appendFileSync

appendFileSync(file, data, options)

- file — полное имя файла или его файловый дескриптор. Принимает значения типа `string`, `FileHandle`. Обязательный параметр.
- data — данные. Принимает значения типа `string`, `ArrayBuffer`, `ArrayBufferView`. Обязательный параметр.
- options (необязательный параметр):
 - writeBOM — запись BOM. Принимает значение типа `boolean`.
 - encoding — кодировка файла. По умолчанию: `'utf8'`.
 - flag — флаг файловой системы. По умолчанию: `'a'`.

Добавляет данные в файл, создавая файл, если он еще не существует. Возвращает `undefined`.

▼ closeSync

closeSync(fd)

- `fd` — файловый дескриптор. Обязательный параметр

Закрывает дескриптор файла. Возвращает `undefined`.

▼ copyFileSync

copyFileSync(src, dest, mode)

- `src` — путь к файлу-источнику операции копирования. Принимает значение типа `string`. Обязательный параметр.
- `dest` — путь к файлу-приемнику операции копирования. Принимает значение типа `string`. Обязательный параметр.
- `mode` — необязательный параметр, определяющий поведение операции копирования. Принимает значение типа `number`. Задается маской, которая может состоять из комбинации следующих значений (например, `constants.COPYFILE_EXCL | constants.COPYFILE_FICLONE`):
 - `constants.COPYFILE_EXCL` — операция копирования завершится ошибкой, если `dest` уже существует.
 - `constants.COPYFILE_FICLONE` — использовать механизм копирования при записи (copy-on-write), при котором физический перенос данных не производится. Если операция не поддерживается, то будет произведено обычное копирование.
 - `constants.COPYFILE_FICLONE_FORCE` — использовать механизм копирования при записи (copy-on-write). Если операция не поддерживается, то копирование завершится с ошибкой.

Копирует **src** в **dest**. По умолчанию **dest** перезаписывается, если он уже существует. Возвращает `undefined`.

▼ existsSync

existsSync(path)

- `path` — путь. Принимает значение типа `string`. Обязательный параметр.

Возвращает `true`, если путь существует, `false` — в противном случае.

▼ fstatSync

fstatSync(fd)

- `fd` — файловый дескриптор. Принимает значение типа `FileHandle`. Обязательный параметр.

Возвращает объект Stats для файлового дескриптора.

▼ ftruncateSync

ftruncateSync(fd, len)

- `fd` — файловый дескриптор. Принимает значение типа `FileHandle`. Обязательный параметр.
- `len` — необязательный параметр, который указывает размер в байтах. Принимает значение типа `number`.

Устанавливает длину обычного файла с файловым дескриптором `fd` в `len` байт. Если файл до этой операции был длиннее, то отсеченные данные теряются. Если файл был короче, то он увеличивается, а добавленная часть заполняется нулевыми байтами. Возвращает `undefined`.

▼ lstatSync

lstatSync(path, options)

- `path` — путь. Принимает значение типа `string`. Обязательный параметр.
- `options` (необязательный параметр):
 - `throwIfNoEntry`. Принимает значение типа `boolean`. Будет ли выдано исключение, если путь в файловой системе не существует. По умолчанию: `true`.

Возвращает объект Stats для `path`. Если `path` — это путь символической ссылки, то возвращает информацию о самой ссылке.

▼ mkdirSync

mkdirSync(path, options, mode)

- `path` — путь. Принимает значение типа `string`. Обязательный параметр.
- `options` (необязательный параметр):
 - `recursive`. Указывает, следует ли создавать родительские каталоги. Принимает значение типа `boolean`. По умолчанию: `false`.
 - `mode`. Под Windows не используется. По умолчанию: `0o777`.

Создает каталог. Возвращает `undefined`.

▼ openSync

openSync(path, flags, mode)

- `path` — путь. Принимает значение типа `string`. Обязательный параметр.
- `flags` — флаги файловой системы. Определяют, в каком режиме открыть файл. Необязательный параметр. По умолчанию: `r`.
- `mode`. Устанавливает права доступа к файлу, когда он создаётся. На Windows влияет на атрибут файла «Только чтение». Подробнее. По умолчанию: `0o666`. Принимает значения типа `number`, `string`.

Возвращает объект, представляющий дескриптор файла.

▼ readdirSync

readdirSync(path, options)

- path — путь. Принимает значение типа `string`. Обязательный параметр.
- options (необязательный параметр):
 - withFileTypes — параметр, определяющий тип возвращаемого значения функции. По умолчанию: `false`.

Читает содержимое каталога. Если `withFileTypes` — `false`, возвращает массив строк. Если `true`, то возвращает массив объектов `Dirent`.

▼ readFileSync

readFileSync(path, options)

- path — путь или файловый дескриптор. Принимает значения типа `string`, `FileHandle`. Обязательный параметр.
- options (необязательный параметр):
 - encoding — кодировка файла. По умолчанию: `'utf8'`.
 - flag — флаг файловой системы. По умолчанию: `'r'`.

Возвращает содержимое файла `path`. Если `encoding` опция указана, то функция возвращает строку, в противном случае возвращает `ArrayBuffer`.

▼ readSync

readSync(fd, buffer, offset, length, position)

- fd — файловый дескриптор. Принимает значение `FileHandle`.
- buffer — буфер, в который записываются данные. Принимает значения `ArrayBuffer`, `ArrayBufferView`. Обязательный параметр.
- offset — позиция `buffer` для записи данных. Необязательный параметр. Принимает значение `number`.
- length — количество байт для чтения. Необязательный параметр. Принимает значение `number`.
- position — указывает, с какого места начать чтение в файле. Если `position` равно `null` или `-1`, то данные будут считаны из текущей позиции в файле, и позиция в файле изменится. Если `position` — целое число, то текущая позиция в файле не изменяется. Необязательный параметр. Принимает значение `number`.

Читает в буфер указанный диапазон байтов из файла. Возвращает число прочитанных байт.

▼ realpathSync

realpathSync(path)

- `path` — относительный путь в файловом хранилище Loginom. Принимает значение типа `string`. Обязательный параметр.

Возвращает абсолютный путь внутри файлового хранилища Loginom.

▼ **renameSync**

renameSync(oldPath, newPath)

- `oldPath` — старый путь. Принимает значение типа `string`. Обязательный параметр.
- `newPath` — новый путь. Принимает значение типа `string`. Обязательный параметр.

Переименовывает файл из `oldPath` в `newPath`. Возвращает `undefined`.

▼ **rmdirSync**

rmdirSync(path, options)

- `path` — путь до каталога. Принимает значение типа `string`. Обязательный параметр.
- `options` (необязательный параметр):
 - `recursive`. Принимает значение типа `boolean`. Если `true`, выполнить рекурсивное удаление каталога. По умолчанию: `false`.

Удаляет папку. Возвращает `undefined`.

▼ **rmSync**

rmSync(path, options)

- `path` — путь. Принимает значение типа `string`. Обязательный параметр.
- `options` (необязательный параметр):
 - `force`. Принимает значение типа `boolean`. Когда `true`, исключение будет игнорироваться, если путь не существует. По умолчанию: `false`.
 - `recursive`. Принимает значение типа `boolean`. Если `true`, выполнить рекурсивное удаление каталога. По умолчанию: `false`.

Удаляет файлы и каталоги. Возвращает `undefined`.

▼ **statSync**

statSync(path, options)

- `path` — путь. Принимает значение типа `string`. Обязательный параметр.
- `options` (необязательный параметр):

- `throwIfNoEntry`. Принимает значение типа `boolean`. Будет ли выдано исключение, если путь в файловой системе не существует. По умолчанию: `true`.

Возвращает объект `Stats` для `path`.

▼ `truncateSync`

`truncateSync(path, len)`

- `path` — путь. Принимает значение типа `string`. Обязательный параметр.
- `len` — необязательный параметр, который указывает размер в байтах. Принимает значение типа `number`.

Устанавливает длину обычного файла с именем `path` в `len` байт. Если файл до этой операции был длиннее, то отсеченные данные теряются. Если файл был короче, то он увеличивается, а добавленная часть заполняется нулевыми байтами. Возвращает `undefined`.

▼ `unlinkSync`

`function unlinkSync(path)`

- `path` — путь. Принимает значение типа `string`. Обязательный параметр.

Удаляет файл. Возвращает `undefined`.

▼ `writeFileSync`

`writeFileSync(path, data, options)`

- `path` — путь или файловый дескриптор. Принимает значения типа `string`, `FileHandle`. Обязательный параметр.
- `data` — содержимое файла. Принимает значения типа `ArrayBuffer`, `ArrayBufferView`. Обязательный параметр.
- `options` (необязательный параметр):
 - `encoding` — кодировка файла. По умолчанию: `'utf8'`.
 - `flag` — флаг файловой системы. По умолчанию: `'r'`.
 - `writeBOM` — запись BOM. Принимает значение типа `boolean`.
 - `mode`. Влияет на права доступа к файлу. [Подробнее](#). Принимает значение типа `number`.

Записывает данные в файл, создавая новый, если он не существует. Возвращает `undefined`.

▼ `writeSync`

`writeSync(fd, buffer, offset, length, position)`

- `fd` — файловый дескриптор (`FileHandle`). Представляет собой открытый файл, в который будут записываться данные.
- `buffer` — буфер данных, из которого будут записаны байты в файл. Обязательный параметр. Может быть объектом типа `ArrayBuffer` или `ArrayBufferView` (например, `Uint8Array` , `Uint16Array` и т.д.).
- `offset` — позиция внутри `buffer` , начиная с которой будут считываться данные для записи. Необязательный параметр. Если не указан, по умолчанию используется начало буфера (`0`). Принимает значение `number` .
- `length` — количество байт, которые нужно записать из `buffer` . Необязательный параметр. Если не указан, записываются все байты от позиции `offset` до конца буфера. Принимает значение `number` .
- `position` — позиция в файле, с которой начнется запись данных. Необязательный параметр. Принимает значение `number` . При значении `null` или отсутствии параметра запись происходит в текущей позиции файла. Отрицательные значения указывают позицию относительно конца файла.

writeSync(fd, string, position, encoding)

- `fd` — файловый дескриптор (`FileHandle`). Представляет собой открытый файл, в который будут записываться данные.
- `string` — строка текста, которая будет записана в файл. Этот параметр является обязательным, если используется данная сигнатура функции. Принимает значение `string` .
- `position` — позиция в файле, с которой начнется запись данных. Необязательный параметр. Принимает значение `number` . При значении `null` или отсутствии параметра запись происходит в текущей позиции файла. Отрицательные значения указывают позицию относительно конца файла.
- `encoding` — определяет кодировку строки, которая будет записана в файл. Необязательный параметр. Принимает значение `Encoding` . Если значение не указано, используется кодировка `utf8` по умолчанию ([список поддерживаемых кодировок](#)).

Записывает `buffer` в указанный файл.

Флаги файловой системы

- `a` : открыть файл для добавления. Файл создается, если он не существует.
- `ax` : такой же, как `a` , но при использовании этого флага возникнет ошибка, если путь существует.
- `a+` : открыть файл для чтения и добавления. Файл создается, если он не существует.
- `ax+` : такой же, как `a+` , но при использовании этого флага возникнет ошибка, если путь существует.
- `as` : открыть файл для чтения и добавления в синхронном режиме. Файл создается, если он не существует. Отключает кэширование данных системой при записи. Данные пишутся сразу на

диск, а не в кэш системы.

- `as+` : открыть файл для добавления в синхронном режиме. Файл создается, если он не существует. Отключает кэширование данных системой при записи. Данные пишутся сразу на диск, а не в кэш системы.
- `r` : открыть файл для чтения. При использовании этого флага возникнет ошибка, если файл не существует.
- `r+` : открыть файл для чтения и записи. При использовании этого флага возникнет ошибка, если файл не существует.
- `rs+` : открыть файл для чтения и записи в синхронном режиме. Отключает кэширование данных системой при записи. Данные пишутся сразу на диск, а не в кэш системы.
- `w` : открыть файл для записи. Файл создается (если он не существует), или удаляется содержимое (если он существует).
- `wx` : такой же, как `w` , но при использовании этого флага возникнет ошибка, если путь существует.
- `w+` : открыть файл для чтения и записи. Файл создается (если он не существует), или удаляется содержимое (если он существует).
- `wx+` : такой же, как `w+` , но при использовании этого флага возникнет ошибка, если путь существует.

Список поддерживаемых кодировок

- `latin1` , `binary` — кодировка **ISO 8859-1, Latin 1**, совпадает с первыми 256 кодовыми позициями Unicode.
- `utf-8` , `utf8` — кодировка **UTF8**.
- `utf16le` , `utf-16le` , `ucs2` , `ucs-2` — кодировка **UTF16 Little-Endian**.
- `ср<кодová страница>` — однобайтовая кодовая страница.
- `iso-8859-<номер>` .

Примеры

Использование File Storage API

Получение списка файлов

```
import * as fs from "builtIn/FS";

let path = "."; // Получить содержимое текущего каталога
let dirents = fs.readdirSync(path, { withFileTypes: true }); // Получить
содержимое каталога path в виде массива объектов Dirent
dirents.sort((e1, e2) => (+e2.isDirectory() - +e1.isDirectory()) ||
(e2.name > e1.name ? -1 : e1.name == e2.name)); // Упорядочить массив
объектов Dirent
for (let i = 0, l = dirents.length; i < l; i++) {
    let dirent = dirents[i];
    let isDir = dirent.isDirectory(),
        name = dirent.name;
    let entry_info = isDir ? `${name}: папка` : `${name}: файл`; //
Проверить - каталог или файл?
    if (dirent.isSymbolicLink()) // Проверить - символическая ссылка?
        entry_info += ", символическая ссылка";
    if (!isDir) {
        let stat = fs.statSync(`${path}/${name}`); // Получить информацию
о файле в виде объекта Stat
        entry_info += `, размер в байтах: ${stat.size}`;
    }
    console.log(entry_info);
}
```

Копирование файла из папки user в папку loginom

```
fs.copyFileSync("/user/test.lgp", "/user/loginom/test.lgp");
```

Перемещение файла из папки user в папку loginom

```
fs.renameSync("/user/test.lgp", "/user/loginom/test.lgp");
```

Удаление файла test.lgp

```
fs.unlinkSync("/user/test.lgp")
```

Получение списка объектов типа Dirent

```
fs.readdirSync("/user/test.lgp", {withFileTypes: true})
```

Усечение файла до 10 первых байт

```
fs.truncateSync("/user/loginom/test.lgp", 10);
```

Копирование папки вместе с содержимым

```
// Импорт модуля для работы с файловой системой
import * as fs from "builtIn/FS";

// Рекурсивная функция для копирования папки и её содержимого
function copyFolderRecursive(srcPath, dstPath) {
    // Чтение содержимого исходной папки (с использованием withFileTypes
    // для получения информации о типе элемента)
    for (const entry of fs.readdirSync(srcPath, { withFileTypes: true }))
    {
        // Имя текущего элемента (файла или папки)
        let name = entry.name;
        // Полный путь к элементу в исходной папке
        let src = srcPath + "/" + name;
        // Полный путь к элементу в целевой папке
        let dst = dstPath + "/" + name;

        if (entry.isFile()) {
            // Если элемент – файл, копируем его в целевую папку
            fs.copyFileSync(src, dst);
        } else if (entry.isDirectory()) {
            // Если элемент – папка, создаем её в целевой папке
            fs.mkdirSync(dst);
            // Рекурсивно копируем содержимое этой папки
            copyFolderRecursive(src, dst);
        }
    }
}

// Основная функция для копирования папки
function copyFolder(srcPath, dstPath) {
    // Проверяем, существует ли целевая папка
    if (!fs.existsSync(dstPath)) {
        // Если целевая папка не существует, создаем её
        fs.mkdirSync(dstPath);
    }
    // Запускаем рекурсивное копирование содержимого исходной папки в
    // целевую
    copyFolderRecursive(srcPath, dstPath);
}
```

```
// Пример использования: копирование папки "/user/deductor" в  
"/user/loginom"  
copyFolder("/user/deductor", "/user/loginom");
```

Запись данных в файл

```
import * as fs from "builtIn/FS";

function str2ab(str) {
    let buf = new ArrayBuffer(str.length * 2); // Создает буфер размером 2
    байта на каждый символ строки
    let bufView = new Uint16Array(buf); // Создает представление Uint16Array
    для ArrayBuffer
    for (let i = 0, strLen = str.length; i < strLen; i++)
        bufView[i] = str.charCodeAt(i); // Записывает код каждого символа в
    буфер
    return buf;
}

var bom_utf16 = new ArrayBuffer(2); // Создаёт ArrayBuffer на 2 байта
var bufView = new Uint16Array(bom_utf16); // Создаём Uint16Array для
работы с буфером
bufView[0] = 0xFEFF; // Записывает BOM (0xFEFF) в буфер

let buf = str2ab("987"); // Преобразует строку «987» в ArrayBuffer

let fd = fs.openSync("new.txt", "w"); // Открывает файл для записи. Если
файла нет, то создает его.
try {
    fs.writeSync(fd, bom_utf16); // Записывает BOM в начало файла
    fs.writeSync(fd, buf, 2, 2); // Записывает часть буфера начиная со 2-го
байта, 2 байта (файл будет содержать строку – 8)
    fs.writeSync(fd, "222", null, "utf16le"); // Записывает строку «222» в
кодировке UTF-16LE (файл будет содержать строку – 8222)
    fs.writeSync(fd, "33", -4, "utf16le"); // Записывает строку «33» в
кодировке UTF-16LE, начиная с позиции -4 (файл будет содержать строку –
8233)
} finally {
    fs.closeSync(fd); // Закрываем файл
}
```

Добавление данных с указанием кодировки

```
fs.appendFileSync('data.txt', 'Привет, мир!', 'utf8');
```

Определение абсолютного пути

```
const realPath = fs.realpathSync(relativeFilePath); //relativeFilePath  
содержит относительный путь
```

Получение информации о файле по дескриптору

```
import * as fs from "builtIn/FS";  
const fd = fs.openSync('example.txt', 'r');  
const size = fs.fstatSync(fd).size; // размер файла в байтах
```