JS Полное описание API

Ниже приведено описание API, специфичного для Loginom и не являющегося частью стандарта языка (ECMA-262).

Глобальное пространство имён

Глобальное свойство `globalThis` содержит значение глобального `this`, который является глобальным объектом.

```
// Объект require применяется для импорта модулей CommonJS
const require: IRequire;

// Представление модуля
interface IModule {
    readonly id: string;              // идентификатор
    parent?: this;                    // вызывающий модуль
    filename?: string;                // полный путь к модулю
    loaded: boolean;                  // загружен ли модуль полностью
    exports: any;                     // экспортируемый объект
}

// Функция require принимает идентификатор модуля и возвращает свойство
модуля exports
interface IRequireFunction {
    (id: string): any;
}

interface IRequire extends IRequireFunction  {
    // функция resolve возвращает полный путь к модулю
    resolve: (id: string) => string;
    // кэш модулей
    cache: { [resolvedId: string]: IModule]; };
}

// Объект console
var console: Console;

interface Console {
     // Метод assert выводит в консоль сообщение, если первый аргумент
false
    assert(condition?: boolean, ...data: any[]): void;
    // Метод error выводит сообщение об ошибке
    error(...data: any[]): void;
    // Методы warn выводит предупреждающее сообщение
    warn(...data: any[]): void;
    // Методы info и log выводят информационное сообщение
    info(...data: any[]): void;
    log(...data: any[]): void;
    // Метод clear очищает консоль вывода сообщений в окне предпросмотра
    clear(): void;
```

```
}
// Функция позволяет отложить выполнение функции
function setTimeout(callback: Function, delay: number = 0, ...args:
any[]): number;

// Функция отменяет таймаут, установленный вызовом setTimeout
function clearTimeout(timeoutID: number): void;

// Функция кодирует в base-64 строку бинарных данных
function btoa(text: string, encoding?: "utf-8"): string;

// Функция декодирует строку, закодированную с использованием base-64
function atob(text: string, encoding?: "utf-8"): string;

// Функция возвращает имя локали текущего узла в формате BCP 47
function getLocale(): string;
```

Встроенный модуль "builtIn/Data"

Объекты модуля "builtIn/Data" предоставляют доступ к портам узла JavaScript. По умолчанию код узла содержит строку импорта этих объектов.

```
const InputTable: IDataSource;      // Источник данных с первого порта
const InputTables: IDataSource[];   // Массив входных источников данных
const OutputTable: IOutputTable;    // Выходной набор данных
const OutputTables: IOutputTable[]; // Массив выходных наборов данных
const InputVariables: IVariables;   // Входные переменные

// Перечисления, описывающие метаданные полей и переменных
enum DataType {
// Тип данных
     None = 0,
     Boolean = 1,
     DateTime = 2,
     Float = 3,
     Integer = 4,
     String = 5,
     Variant = 6
}

enum DataKind {
// Вид данных
     Undefined = 0,
     Continuous = 1,
     Discrete = 2
}

enum UsageType {
// Назначение полей
     Unspecified = 0,
     Excluded = 1,
     Useless = 2,
     Used = 3,
     Input = 3,
     Active = 3,
     Output = 4,
     Predicted = 4,
     Key = 5,
     Group = 6,
     Value = 7,
     Transaction = 8,
     Item = 9
}
```

```
// Информация о столбце
interface IColumnInfo {
    readonly Name: string;
// Имя
    readonly DisplayName: string;
// Метка
    readonly DataType: DataType;
// Тип данных
    readonly DataKind: DataKind;
// Вид данных
    readonly DefaultUsageType: UsageType;
// Назначение поля
}

// Представление столбца набора данных
interface IColumn extends IColumnInfo, Iterable<boolean | number | string
| Date | undefined> {
    readonly Index: number;
// Индекс
    readonly RowCount: number;
// Количество значений
    // Метод Get возвращает значение столбца по индексу
    Get(row: number): boolean | number | string | Date | undefined;
    // Метод IsNull проверяет на Null значение столбца
    IsNull(row: number): boolean;
}

// Свойства столбца входного набора данных
interface IIntputColumn extends IColumn {
    readonly UsageType: UsageType;
// Назначение поля
}

// Свойства и методы столбца выходного набора данных
interface IOutputColumn extends IColumn {
    // Свойства столбца доступны на чтение и запись
    DisplayName: string;
// Метка
    DataType: DataType;
// Тип данных
    DataKind: DataKind;
```

```
// Вид данных
    DefaultUsageType: UsageType;
// Назначение поля
    // Метод Set задает значение поля в записи, созданной методом Append
    Set(value: boolean | number | string | Date | null | undefined): void;
}

// Доступ к итерируемому списку столбцов по имени и индексу
interface IIntputColumns extends Iterable<IIntputColumn> {
    [name: string]: IIntputColumn;
    [index: number]: IIntputColumn;
}

// Свойства и методы набора данных
interface IDataSource {
    readonly Columns: IIntputColumns;
// Список столбцов
    readonly ColumnCount: number;
// Количество столбцов
    readonly RowCount: number;
// Количество строк

    // Метод Get возвращает значение заданной строки в заданном столбце
    Get(row: number, col: number | string): boolean | number | string |
Date | undefined;
    // Метод IsNull проверяет на Null значение заданной строки в заданном
столбце
    IsNull(row: number, col: number | string): boolean;
    // Метод GetColumn возвращает столбец источника данных
    GetColumn(col: number | string): IIntputColumn;
}

// Свойства и методы выходного набора
interface IOutputTable extends IDataSource {
    readonly Columns: IOutputColumns;
// Список столбцов
    // Метод AssignColumns создает столбцы из итерируемого источника
    AssignColumns(source: Iteratable<string | IColumnInfo>): void;
    // Метод GetColumn возвращает столбец выходного набора
    GetColumn(col: number | string): IOutputColumn;
    // Метод AddColumn добавляет столбец в конец списка столбцов
    AddColumn(source?: string | IColumnInfo): IOutputColumn;
```

```
    // Метод InsertColumn вставляет столбец по заданному индексу
    InsertColumn(col: number, source?: IColumnInfo): IOutputColumn;
    // Метод DeleteColumn удаляет столбец по индексу или имени
    DeleteColumn(col: number | string): void;
    // Метод ClearColumns удаляет все столбцы
    ClearColumns(): void;
    // Метод Append добавляет запись в набор
    Append(): void;
    // Метод Set задает значение заданного поля в записи, созданной
методом Append
    Set(col: number | string, value: boolean | number | string | Date |
null | undefined): void;
}

// Доступ к итерируемому списку столбцов выходного набора по имени и
индексу
interface IOutputColumns extends Iterable<IOutputColumn> {
    [index: number]: IOutputColumn;
    [name: string]: IOutputColumn;
}

// Представление входной переменной
interface IVariable {
    readonly Index: number;
// Индекс
    readonly Name: string;
// Имя
    readonly DisplayName: string;
// Метка
    readonly DataType: DataType;
// Тип данных
    readonly Value: boolean | number | string | Date | undefined;
// Значение
    readonly IsNull: boolean;
// Проверка на Null
}

// Доступ к итерируемому списку входных переменных по имени и индексу
interface IVariableItems extends Iterable<IVariable> {
    [name: string]: IVariable;
    [index: number]: IVariable;
}
```

```
// Представление переменных входного порта
interface IVariables {
    readonly Items: IVariableItems;
    readonly Count: number;
}
```

Встроенный модуль "builtIn/Fetch"

`Fetch API` — интерфейс для работы с запросами и ответами HTTP, предоставляет возможность взаимодействия с ресурсами сети непосредственно из узла JavaScript.

```
// Представление заголовков запроса и ответа
interface Headers {
    append(name: string, value: string): void;
    delete(name: string): void;
    get(name: string): string | null;
    has(name: string): boolean;
    set(name: string, value: string): void;
    forEach(callbackfn: (value: string, key: string, parent: Headers) =>
void, thisArg?: any): void;
    [Symbol.iterator](): IterableIterator<[string, string]>;
    entries(): IterableIterator<[string, string]>;
    keys(): IterableIterator<string>;
    values(): IterableIterator<string>;
}
type HeadersInit = Headers | IterableIterator<[string, string]> |
Record<string, string>;
declare var Headers: {
    prototype: Headers;
    new(init?: HeadersInit): Headers;
}

// Представление тела запроса/ответа
interface Body {
    readonly bodyUsed: boolean;
    arrayBuffer(): Promise<ArrayBuffer>;
    json(): Promise<any>;
    text(): Promise<string>;
}

// Представление ошибки отмены без явно указанной причины
interface AbortError extends Error {
    name: "AbortError";
    message: "signal is aborted without reason";
}

// Представление сигнала отмены
interface AbortSignal {
    readonly aborted: boolean;
    readonly reason: any;
    onabort: ((this: AbortSignal, ev: {type: "abort"; reason: any}) =>
any) | null | undefined;
```

```
    addEventListener(type: "abort", listener: (this: AbortSignal, ev:
{type: "abort"; reason: any}) => any): void;
    removeEventListener(type: "abort", listener: (this: AbortSignal, ev:
{type: "abort"; reason: any}) => any): void;
}

// Представление контроллера отмены
interface AbortController {
    readonly signal: AbortSignal;
    abort(reason?: any): void;
}
declare var AbortController: {
    prototype: AbortController;
    new(): AbortController;
}

// Представление запроса
type RequestInfo = Request | string;
type RequestRedirect = "error" | "follow" | "manual";
interface Request extends Body {
    readonly headers: Headers;
    readonly method: string;
    readonly redirect: RequestRedirect;
    readonly signal: AbortSignal;
    readonly url: string;
    clone(): Request;
}
type BodyInit = ArrayBufferView | ArrayBuffer | string;
interface RequestInit {
    body?: BodyInit | null;
    headers?: HeadersInit;
    method?: string;
    redirect?: RequestRedirect;
    signal?: AbortSignal;
}
declare var Request: {
    prototype: Request;
    new(input: RequestInfo, init?: RequestInit): Request;
}

// Представление ответа
interface Response extends Body {
```

```
    readonly headers: Headers;
    readonly ok: boolean;
    readonly redirected: boolean;
    readonly status: number;
    readonly statusText: string;
    readonly url: string;
    clone(): Response;
}
interface ResponseInit {
    headers?: HeadersInit;
    status?: number;
    statusText?: string;
}
declare var Response: {
    prototype: Response;
    new(body?: BodyInit | null, init?: ResponseInit): Response;
    error(): Response;
    redirect(url: string, status?: number): Response;
}

// Функция запроса ресурса сети
function fetch(url: Request|string, init?: RequestInit):
Promise<Response>;
```

Встроенный модуль "builtIn/FS"

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

```
namespace constants {
    COPYFILE_EXCL: number;
    COPYFILE_FICLONE: number;
    COPYFILE_FICLONE_FORCE: number;
    O_RDONLY: number;
    O_WRONLY: number;
    O_RDWR: number;
    O_CREAT: number;
    O_EXCL: number;
    O_TRUNC: number;
    O_APPEND: number;
    O_SYNC: number;
    S_IFMT: number;
    S_IFREG: number;
    S_IFDIR: number;
    S_IFLNK: number;
}

type Encoding = "utf8" | "utf-8" | "utf16le" | "ucs2" | "ucs-2" | "latin1"
| "binary";
// или явное указание номера кодовой страницы однобайтовой кодировки в
формате cp<CodePageNumber>, например, cp1252
// или явное указание номера кодовой страницы iso-8859 в формате "iso-
8859-<номер>", например, iso-8859-1
type OpenMode = number | string;
type Mode = number | string;

class Stats {
    isFile(): boolean;
    isDirectory(): boolean;
    isSymbolicLink(): boolean;

    mode: number;
    size: number;
    atime: Date;
    mtime: Date;
    ctime: Date;
    birthtime: Date;
}
```

```
class FileHandle {
    valueOf(): number;
}

class Dirent {
    isFile(): boolean;
    isDirectory(): boolean;
    isSymbolicLink(): boolean;
    name: string;
}

interface ReadSyncOptions {
   offset?: number;
   length?: number;
   position?: number | null;
}

interface WriteFileOptions {
  encoding?: string;
  flag?: string;
  writeBOM?: boolean;
}

function appendFileSync(file: string | FileHandle, data: string |
ArrayBuffer | ArrayBufferView, options?: WriteFileOptions): void;

function closeSync(fd: FileHandle): void;

function copyFileSync(src: string, dest: string, mode?: number): void;

function existsSync(path: string): boolean;

function fstatSync(fd: FileHandle): Stats;

function ftruncateSync(fd: FileHandle, len?: number | null): void;

function lstatSync(path: string, options?: { throwIfNoEntry?: boolean }):
Stats;

function mkdirSync(path: string, options?: { recursive?: boolean; mode?:
Mode } | Mode): void;
```

```
function openSync(path: string, flags: OpenMode; mode: Mode): FileHandle;

function readdirSync(path: string, options?: { withFileTypes?: false }):
string[];
function readdirSync(path: string, options: { withFileTypes: true }):
Dirent[];

function readFileSync(path: string | FileHandle, options?: { encoding?:
null; flag?: string; } | null): ArrayBuffer;
function readFileSync(path: string | FileHandle, options: { encoding:
Encoding; flag?: string; } | Encoding): string;

function readSync(fd: FileHandle, buffer: ArrayBuffer | ArrayBufferView,
offset?: number | null, length?: number | null, position?: number | null):
number;
function readSync(fd: FileHandle, buffer: ArrayBuffer | ArrayBufferView,
opts?: ReadSyncOptions): number;

function realpathSync(path: string): string;

function renameSync(oldPath: string, newPath: string): void;

function rmdirSync(path: string, options?: { recursive?: boolean }): void;

function rmSync(path: string, options?: { force?: boolean; recursive?:
boolean }): void;

function statSync(path: string, options?: { throwIfNoEntry?: boolean }):
Stats;

function truncateSync(path: string, len?: number | null): void;

function unlinkSync(path: string): void;

function writeFileSync(path: string | FileHandle, data: string |
ArrayBuffer | ArrayBufferView, options?: WriteFileOptions): void;

function writeSync(fd: FileHandle, buffer: ArrayBuffer | ArrayBufferView,
offset?: number | null, length?: number | null, position?: number | null):
number;
```

```
function writeSync(fd: FileHandle, string: string, position?: number |
null, encoding?: Encoding): number;
```