

Max 9 JS API

Version 9.2.0-rev.1

Cycling ’74

cycling74.com

Contents

Max 9 JS API 5
class AbortController 13
class AbortSignal 15
interface AddressInfo 18
interface AttributeOptions 20
type Basic2dStrokeStyleParameterNames 24
class Blob 25
type BodyInit 30
module buffer 31
class Buffer 40
interface Console 49
module dgram 56
class Dict 66
type DrawingPrimitiveType 81
class EventSource 82
interface EventSourceMessageEvent 88
class File 90
type FileAccess 105
type FileEndianness 106
type FileFourCharacterCode 107
type FileLineEndingStyle 0
class Folder 0
class FormData 0
function clearImmediate 0
function clearInterval 0
function clearTimeout 0
consoleConsole 0
function cpost 0
function error 0
function fetch 0
globalThisjsthis 0
Jitter3DUtilsJitter3dUtilsInterface 0
maxMax 0
function messnamed 0
mgraphicsMGraphics 0

function post	0
function queueMicrotask	0
function setImmediate	0
function setInterval	0
function setTimeout	0
sketchSketch	0
class Global	0
class Headers	0
type HeadersInit	0
module http	0
module https	0
class Image	0
interface Jitter3dUtilsInterface	0
type Jitter3dUtilsMat16	0
type Jitter3dUtilsVec3	0
type Jitter3dUtilsVec4	0
interface JitterEvent	0
type JitterEventType	0
class JitterListener	0
class JitterMatrix	0
class JitterObject	0
interface jsthis	0
type LineStrokeStyleParameterNames	0
class LiveAPI	0
interface Max	0
class MaxArray	0
class MaxFFT	0
class MaxFFT2D	0
interface MaxFFT2DOptions	0
interface MaxFFTOptions	0
class Maxobj	0
interface MaxobjConnection	0
class MaxobjListener	0
interface MaxobjListenerData	0
class MaxString	0
class MGraphics	0
type MGraphicsMatrixHandle	0
type MGraphicsPathHandle	0
class MGraphicsSVG	0
module net	0
interface NetworkError	0

class ParameterInfoProvider 0
interface ParameterInfoProviderData 0
class ParameterListener 0
interface ParameterListenerData 0
class Patcher 0
interface Pattern 0
interface PointerEvent 0
class PolyBuffer 0
interface ProgressEvent 0
class ReadableStream 0
interface ReadableStreamDefaultReader 0
interface ReadableStreamReadResult 0
class Request 0
type RequestCredentials 0
interface RequestInit 0
type RequestRedirect 0
class Response 0
class Rx256 0
class Sketch 0
class SnapshotAPI 0
class SQLite 0
class SQLResult 0
class Task 0
class TextDecoder 0
class TextEncoder 0
module tls 0
class TransformStream 0
class URL 0
class URLSearchParams 0
class WebSocket 0
interface WebSocketCloseEvent 0
interface WebSocketConnection 0
interface WebSocketErrorEvent 0
interface WebSocketMessageEvent 0
class WebSocketServer 0
interface Wind 0
class WritableStream 0
interface WritableStreamDefaultWriter 0
class XMLHttpRequest 0
interface XMLHttpRequestUpload 0
Credits 0

Max 9 JS API

Globals 5
Functions 6
Built-In Modules 7
Classes 7
Interfaces 9
Type Aliases 11

Reference for the API available within the `[js]`, `[jsui]`, `[v8]`, `[v8.codebox]`, and `[v8ui]` objects

Some classes and interfaces are only available with the v8 engine. For more information, refer to the JavaScript Documentation

Globals

Item	Engine	Description
console	v8	The global `console` instance, the Max-console-backed `console` object exposed by the v8 runtime.
globalThis		The "this" object in the context of JavaScript in Max. The object is the "this" within the context of any function you define that can be invoked from Max, as well as in your global code. See jsthis for more details.
Jitter3DUtils		The global `Jitter3DUtils` interface, providing utility functions for 3D vector and matrix operations. See Jitter3dUtilsInterface for more details.

max	Singleton Max object to access and control the Max environment. See Max for more details. In addition to the properties and methods described here, you can also use the global Max object to call methods as defined in Controlling Max with Messages
mgraphics	The global `mgraphics` instance, for rendering shapes and images in custom Max objects.
sketch	The global `sketch` instance, for drawing in custom Max objects with an OpenGL-backed drawing context.

Functions

Item	Engine	Description
clearImmediate	v8	Cancels an immediate previously created with setImmediate().
clearInterval	v8	Cancels an interval previously created with setInterval().
clearTimeout	v8	Cancels a timeout previously created with setTimeout().
cpost		Prints a message to the system console window. See post() for more details about arguments and formatting.
error		Prints a message to the Max console with a red tint. See post() for more details about arguments and formatting.
fetch	v8	Fetches a resource over HTTP or HTTPS.
messnamed		Sends a message to the named Max object.
post		Prints a representation of the arguments in the Max window.
queueMicrotask	v8	Queues a microtask to run after the current task completes and before the next timer callback.
setImmediate	v8	Schedules a callback to run on the next scheduler tick.
setInterval	v8	Schedules a callback to run repeatedly at a fixed interval.
setTimeout	v8	Schedules a one-shot callback to run after a delay.

Built-In Modules

Item	Engine	Description
buffer	v8	The `buffer` module exposed by `require("buffer")`, providing the buffer.IOBuffer class for handling binary data.
dgram	v8	The `dgram` module exposed by `require("dgram")`.
http	v8	The `http` module exposed by `require("http")` to access HTTP/1.1 clients and servers. This is a minimal `http` subset, not full Node parity.
https	v8	The `https` module exposed by `require("https")`. Mirrors http over TLS, defaulting to `https:` URLs.
net	v8	The `net` module exposed by `require("net")` providing TCP client and server sockets.
tls	v8	The `tls` module exposed by `require("tls")`: TLS/SSL client and server sockets.

Classes

Item	Engine	Description
AbortController	v8	Creates an AbortSignal and the means to abort it.
AbortSignal		Signals abortion to abortable operations such as fetch().
Blob	v8	An immutable blob of binary data with a MIME type.
Buffer		Bind to a Max `buffer~` object.
Dict		Bind to a Max `dict` object.
EventSource	v8	A Server-Sent Events client.

File		The File object provides a means of reading and writing files from JavaScript.
Folder		Iterate through the files in a folder.
FormData	v8	A `multipart/form-data` field set.
Global		Global object for sharing data between Max JavaScript instances.
Headers	v8	A case-insensitive multimap of HTTP headers.
Image		Bitmap image object handle
JitterListener		A listener for changes in a JitterObject.
JitterMatrix		A named matrix which may be used for data storage and retrieval, resampling, and matrix type and planecount conversion operations.
JitterObject		A JavaScript representation of a Jitter object in a patcher.
LiveAPI		A means of communicating with the Live API from JavaScript.
MaxArray		Bind a Max `array` object
MaxFFT		Fast Fourier Transform processor
MaxFFT2D		Two-dimensional Fast Fourier Transform processor
Maxobj		A JavaScript representation of a Max object in a patcher.
MaxobjListener		A listener for changes in a Maxobj object.
MaxString		Bind a Max `string` object
MGraphics		Drawing context for rendering shapes and images.
MGraphicsSVG		SVG object handle
ParameterInfoProvider		Provides a list of named parameter objects within a patcher hierarchy as well as information about specific parameter objects. It can also notify when parameter objects are added or removed from a patcher hierarchy.
ParameterListener		A listener for changes in named parameters.

Patcher		A JavaScript representation of a Max patcher.
PolyBuffer		Bind to a Max `polybuffer~` object.
ReadableStream	v8	A byte-oriented readable stream. Used for Response.body and as a streamed request body.
Request	v8	A request to be sent with fetch().
Response	v8	A response returned by fetch().
Rx256		Xoxhiro256+ random number generator
Sketch		Interface to an OpenGL-backed drawing context
SnapshotAPI		Provides access to patcher snapshots.
SQLite		Provides access to the SQLite database system.
SQLResult		A container for results obtained in an SQLite.exec() call.
Task		A function that can be scheduled or repeated.
TextDecoder	v8	Decodes UTF-8 bytes into a JavaScript string.
TextEncoder	v8	Encodes JavaScript strings into UTF-8 bytes.
TransformStream	v8	A byte-oriented transform stream pairing a writable input with a readable output.
URL	v8	Parses and manipulates URLs.
URLSearchParams	v8	Reads and manipulates URL query strings.
WebSocket	v8	A WebSocket client.
WebSocketServer	v8	A WebSocket server.
WritableStream	v8	A byte-oriented writable stream companion for piping.
XMLHttpRequest		XMLHttpRequest provides HTTP client functionality for making network requests from JavaScript in Max.

Interfaces

Item	Engine	Description
AddressInfo		The local or remote endpoint of a socket, as returned by `address()`.
AttributeOptions		Options accepted by the v8-only forms of declareattribute, describing how an attribute is typed, stored, and presented.
Console		The Max-console-backed `console` object exposed by the v8 runtime. Output is routed to the Max console: `log`, `info`, and `debug` use normal post rows, `warn` uses warning rows, and `error`, `assert`, and `trace` use error rows. Formatting is a pragmatic subset: string joining, `Error` stack/message rendering, JSON-ish object rendering with a circular fallback, and the `%s`, `%d`, `%i`, `%f`, `%j`, `%o`, `%O`, and `%%` format tokens. Full Node `util.inspect`, `console.table`, grouping, profiling, and configurable output streams are not implemented.
EventSourceMessageEvent		An event delivered to an EventSource.
Jitter3dUtilsInterface		Utilities for Jitter 3D manipulations
JitterEvent		The argument passed to a JitterListener callback function.
jsthis		The "this" object in the context of JavaScript in Max
Max		Interface representing the current Max environment providing access and control.
MaxFFT2DOptions		Options for constructing a MaxFFT2D instance.
MaxFFTOptions		Options for constructing a MaxFFT instance.
MaxobjConnection		A JavaScript representation of a patchcord connection.
MaxobjListenerData		The argument provided to a MaxobjListener callback function.
NetworkError		Networking Error The shape of `Error` objects delivered to networking

	`'error'` callbacks and promise rejections across fetch(), the `net` / `tls` / `dgram` modules, the `http` / `https` modules, WebSocket, and EventSource.
ParameterInfoProviderData	The argument to the ParameterInfoProvider's callback function
ParameterListenerData	The argument provided to a ParameterListener callback function.
Pattern	Pattern object for drawing gradients and patterns.
PointerEvent	Pointer event object passed to onpointer* event handlers
ProgressEvent	ProgressEvent provides information about the progress of a network request.
ReadableStreamDefaultReader	A reader that pulls chunks from a ReadableStream.
ReadableStreamReadResult	The result of a single ReadableStreamDefaultReader.read() call.
RequestInit	Options accepted by fetch() and the Request constructor.
WebSocketCloseEvent	A `close` event delivered to a WebSocket.
WebSocketConnection	A server-side WebSocket connection accepted by a WebSocketServer.
WebSocketErrorEvent	An `error` event delivered to a WebSocket.
WebSocketMessageEvent	A `message` event delivered to a WebSocket.
Wind	A property of the Patcher which represents its window.
WritableStreamDefaultWriter	A writer that pushes chunks to a WritableStream.
XMLHttpRequestUpload v8	The upload object of an XMLHttpRequest, exposing progress events for the request body.

Type Aliases

Item	Description
Basic2dStrokeStyleParameterNames	Stroke parameters for use with Sketch.beginstroke() in the "basic2d" drawing style
BodyInit	Acceptable values for an outgoing request or response body.
DrawingPrimitiveType	Primitive type to use for drawing shapes. See Sketch.shapeprim().
FileAccess	File access type Used with File
FileEndianness	File endianness Used with File
FileFourCharacterCode	Four character file codes. Used with File
FileLineEndingStyle	File line ending style Used with File
HeadersInit	Acceptable initializers for Headers.
Jitter3dUtilsMat16	A 4x4 matrix Used with Jitter3dUtilsInterface
Jitter3dUtilsVec3	A 3D vector Used with Jitter3dUtilsInterface
Jitter3dUtilsVec4	A 4D vector Used with Jitter3dUtilsInterface
JitterEventType	Event types Possible event types for a JitterEvent
LineStrokeStyleParameterNames	Stroke parameters for use with Sketch.beginstroke() in the "line" drawing style
MGraphicsMatrixHandle	Opaque handle for an MGraphics transformation matrix.
MGraphicsPathHandle	Opaque handle for an MGraphics path.
RequestCredentials	Credentials mode controlling cookie sending/storage for a request.
RequestRedirect	Redirect-following mode for a request.

class AbortController

Index 13
new AbortController() 14
abort(reason) 14
signal 14

Supported only in the `v8` *engine. Introduced in Max v9.2.0.*

Creates an AbortSignal and the means to abort it.

Example

```
const controller = new AbortController();
setTimeout(() => controller.abort(), 1000);
fetch("https://example.com/slow", { signal: controller.signal
}).catch((err) =>
  post("aborted:", err.message, "\n"),
);
```

Index

Constructors

- new AbortController()

Properties

- signal

Methods

- abort(reason)

new AbortController()

```
new AbortController();
```

Constructs a new instance of the `AbortController` class

abort(reason)

Aborts associated operations and marks AbortController.signal as aborted.

```
abort(reason?: any): void;
```

Name	Type	Description
optional reason	any	an optional abort reason

signal AbortSignal read-only

The signal to pass to abortable operations.

class AbortSignal

Index 15
aborted 15
addEventListener(type, listener) 16
reason 16
removeEventListener(type, listener) 16
AbortSignal.timeout(ms) 17

Signals abortion to abortable operations such as fetch().

Index

Properties

- aborted
- reason

Methods

- addEventListener(type, listener)
- removeEventListener(type, listener)
- AbortSignal.timeout(ms)

aborted boolean read-only

Whether the signal has been aborted.

addEventListener(type, listener)

Registers a listener.

```
addEventListener(type: string, listener: (...args: any[]) => void): void;
```

Name	Type	Description
type	string	the event name, typically `"abort"`
listener	(...args: any[]) => void	the callback to invoke

reason any read-only

The abort reason, when AbortSignal.aborted is true.

removeEventListener(type, listener)

Removes a previously registered listener.

```
removeEventListener(type: string, listener: (...args: any[]) => void):
void;
```

Name	Type	Description
type	string	the event name

listener	(...args: any[]) => void	the callback to remove

AbortSignal.timeout(ms) static

Returns a signal that aborts automatically after `ms` milliseconds.

```
static timeout(ms: number): AbortSignal;
```

Name	Type	Description
ms	number	the timeout in milliseconds
Return Value	AbortSignal	

interface AddressInfo

Index 18
address 18
family 18
port 19

The local or remote endpoint of a socket, as returned by `address()`.

IPC endpoints report `family: "IPC"` and `port: 0`, with `address` set to the socket path or pipe name. On Windows, both the connecting and accepted ends report the pipe name; on POSIX the connecting/accepted end reports `<unnamed>`, because Unix client sockets are not bound to a path.

Index

Properties

- address
- family
- port

address string

family string

port number

interface AttributeOptions

Index 20
category 21
default 21
embed 21
enumindex 22
enumvals 22
getter 22
invisible 22
label 22
max 22
min 22
name 23
paint 23
setter 23
size 23
steps 23
style 23
type 23

Options accepted by the v8-only forms of declareattribute, describing how an attribute is typed, stored, and presented.

Index

Properties

- category
- default

- embed
- enumindex
- enumvals
- getter
- invisible
- label
- max
- min
- name
- paint
- setter
- size
- steps
- style
- type

category string

The attribute's category.

default number | string | (number | string)[]

The default value.

embed number | boolean

Whether to embed the attribute in the patcher file on save.

enumindex string[]

enumvals string[]

getter string

The getter function name.

invisible number | boolean

Whether the attribute is invisible.

label string

A human-readable label.

max number

The maximum value.

min number

The minimum value.

name string

The attribute name. Required when `declareattribute` is called with a single options object.

paint number | boolean

setter string

The setter function name.

size number

The array size, for array-valued attributes (implicit if `default` is an array).

steps number

The number of steps.

style "onoff" | "enum" | "rgba"

The attribute's editor style.

type "long" | "float" | "symbol" | "atom"

The attribute's data type.

type Basic2dStrokeStyleParameterNames

Stroke parameters for use with Sketch.beginstroke() in the "basic2d" drawing style

- `"alpha"`: May vary point to point. Value is specified as an alpha value. Useful if alpha is the only color channel which will vary throughout the path. - `"color"`: May vary point to point. Values are specified as red, green, blue, alpha. - `"order"`: Global for a given path. Value must be interpolation order. Default is 3, or bi-cubic interpolation. - `"outline"`: Global for a given path. Value is 0 (off) or 1 (on). Default is 1. - `"outcolor"`: May vary point to point. Outline color. Values are specified as red, green, blue, and alpha values. If no color is specified, then the outline color will be the same as the interior color. - `"scale"`: May vary point to point. Value is specified as an width value. Width of the stroked path. - `"slices"`: Global for a given path. Number of slices for a curved section. Default is 20.

```
export declare type Basic2dStrokeStyleParameterNames =
  "alpha" | "color" | "order" | "outline" | "outcolor" | "scale" |
"slices";
```

class Blob

Index 25
new Blob(parts, options) 26
arrayBuffer() 26
bytes() 27
size 27
slice(start, end, contentType) 27
stream() 28
text() 28
type 29

Supported only in the `v8` *engine. Introduced in Max v9.2.0.*

An immutable blob of binary data with a MIME type.

Index

Constructors

- new Blob(parts, options)

Properties

- size
- type

Methods

- arrayBuffer()
- bytes()
- slice(start, end, contentType)
- stream()
- text()

new Blob(parts, options)

```
new Blob(
  parts?: Array<string | ArrayBuffer | ArrayBufferView | Blob>,
  options?: { type?: string },
);
```

Constructs a new instance of the `Blob` class

Parameter	Type	Description
optional parts	Array<string | ArrayBuffer | ArrayBufferView | Blob>	the data parts that make up the blob.
optional options	object	blob options; `type` sets the MIME type. *optional* options.type string

arrayBuffer()

Reads the blob contents as an `ArrayBuffer`.

```
arrayBuffer(): Promise<ArrayBuffer>;
```

Name	Type	Description
Return Value	Promise<ArrayBuffer>	

bytes()

Reads the blob contents as bytes.

```
bytes(): Promise<buffer.IOBuffer>;
```

Name	Type	Description
Return Value	Promise<buffer.IOBuffer>	

size number read-only

Size of the blob in bytes.

slice(start, end, contentType)

Returns a new blob containing a byte range of this blob.

```
slice(start?: number, end?: number, contentType?: string): Blob;
```

Name	Type	Description
optional start	number	start byte offset
optional end	number	end byte offset (exclusive)
optional contentType	string	MIME type for the new blob
Return Value	Blob	

stream()

Returns a byte-oriented ReadableStream over the blob contents.

```
stream(): ReadableStream;
```

Name	Type	Description
Return Value	ReadableStream	

text()

Reads the blob contents as a UTF-8 string.

```
text(): Promise<string>;
```

Name	Type	Description
Return Value	Promise<string>	

type string read-only

MIME type of the blob, or `""`.

type BodyInit

Acceptable values for an outgoing request or response body.

Buffered bodies accept strings, byte views, `Blob`, `FormData`, and `URLSearchParams`. A ReadableStream body is streamed and requires `duplex: "half"` on the request init.

```
export declare type BodyInit =
  | string
  | buffer.IOBuffer
  | ArrayBuffer
  | ArrayBufferView
  | Blob
  | FormData
  | URLSearchParams
  | ReadableStream;
```

module buffer

Index 32
class IOBuffer 32
new IOBuffer(value, encoding) 33
IOBuffer.alloc(size, fill) 34
IOBuffer.byteLength(value, encoding) 35
IOBuffer.concat(items) 35
equals(other) 35
IOBuffer.from(value, encoding) 36
IOBuffer.from(arrayLike, mapfn, thisArg) 36
IOBuffer.from(arrayLike, mapfn, thisArg) 37
IOBuffer.isBuffer(value) 38
slice(start, end) 38
toString(encoding, start, end) 38
toUint8Array() 39
type IOBufferEncoding 39

Supported only in the `v8` engine. Introduced in Max v9.2.0.

The `buffer` module exposed by `require("buffer")`, providing the buffer.IOBuffer class for handling binary data.

`IOBuffer` is the binary type used throughout the v8 networking APIs. It is a `Uint8Array` subclass in the spirit of Node's `Buffer`, but is not full Node `Buffer` parity. There is no global `Buffer` alias (which would conflict with the Max `buffer~` wrapper). See buffer.IOBuffer for details.

Example

```
const { IOBuffer } = require("buffer");
const buf = IOBuffer.from("hello");
post(buf.toString(), "\n"); // hello
```

Index

Classes

- IOBuffer

Type Aliases

- IOBufferEncoding

class IOBuffer

A `Uint8Array` subclass for binary data, used throughout the v8 networking APIs.

`IOBuffer` is the binary type used throughout the networking APIs. It is a subclass of `Uint8Array`. There is no global `Buffer` alias (which would conflict with the Max `buffer~` wrapper), and `IOBuffer` is not full Node `Buffer` parity.

Obtain the class with `const { IOBuffer } = require("buffer")`.

Example

```
const { IOBuffer } = require("buffer");
const a = IOBuffer.from("hello");
const b = IOBuffer.from("68656c6c6f", "hex");
post(a.equals(b), "\n"); // true
post(a.toString("base64"), "\n");
```

Constructors

- new IOBuffer(value, encoding)

Methods

- IOBuffer.alloc(size, fill)
- IOBuffer.byteLength(value, encoding)
- IOBuffer.concat(items)
- equals(other)
- IOBuffer.from(value, encoding)
- IOBuffer.from(arrayLike, mapfn, thisArg)
- IOBuffer.from(arrayLike, mapfn, thisArg)
- IOBuffer.isBuffer(value)
- slice(start, end)
- toString(encoding, start, end)
- toUint8Array()

new IOBuffer(value, encoding)

```
new IOBuffer(
  value?:
    number | ArrayBuffer | ArrayBufferView | string | ArrayLike<number>,
  encoding?: IOBufferEncoding,
);
```

Constructs a new instance of the `IOBuffer` class

Parameter	Type	Description
optional value	number | ArrayBuffer | ArrayBufferView | string | ArrayLike<number>	a byte length, an `ArrayBuffer`, a typed array / view, a string (decoded with `encoding`), or an array-like of byte values.
optional encoding	IOBufferEncoding	the encoding used when `value` is a string (default `"utf8"`).

IOBuffer.alloc(size, fill)

static

Allocates a zero- or fill-initialized `IOBuffer` of the given size.

```
static alloc(size: number, fill?: number | string): IOBuffer;
```

Name	Type	Description
size	number	
optional fill	number | string	
Return Value	IOBuffer	

IOBuffer.byteLength(value, encoding)

static

Returns the byte length of a string in the given encoding.

```
static byteLength(value: string, encoding?: IOBufferEncoding): number;
```

Name	Type	Description
value	string	
optional encoding	IOBufferEncoding	
Return Value	number	

IOBuffer.concat(items)

static

Concatenates several buffers into one.

```
static concat(items: IOBuffer[]): IOBuffer;
```

Name	Type	Description
items	IOBuffer[]	
Return Value	IOBuffer	

equals(other)

Returns whether this buffer has the same bytes as another.

```
equals(other: IOBuffer): boolean;
```

Name	Type	Description
other	IOBuffer	
Return Value	boolean	

IOBuffer.from(value, encoding)

static

Creates an `IOBuffer` from a value.

```
static from(
  value: ArrayBuffer | ArrayBufferView | string | ArrayLike<number>,
  encoding?: IOBufferEncoding,
): IOBuffer;
```

Name	Type	Description
value	ArrayBuffer | ArrayBufferView | string | ArrayLike<number>	
optional encoding	IOBufferEncoding	
Return Value	IOBuffer	

IOBuffer.from(arrayLike, mapfn, thisArg)

static

Inherited `Uint8Array.from` form with a map function.

```
static from(
  arrayLike: Iterable<number> | ArrayLike<number>,
  mapfn?: (v: number, k: number) => number,
  thisArg?: any,
): IOBuffer;
```

Name	Type	Description
arrayLike	Iterable<number> | ArrayLike<number>	
optional mapfn	(v: number, k: number) => number	
optional thisArg	any	
Return Value	IOBuffer	

IOBuffer.from(arrayLike, mapfn, thisArg)

static

Inherited `Uint8Array.from` form with a map function.

```
static from<T>(
  arrayLike: Iterable<T> | ArrayLike<T>,
  mapfn: (v: T, k: number) => number,
  thisArg?: any,
): IOBuffer;
```

Name	Type	Description
arrayLike	Iterable<T> | ArrayLike<T>	
mapfn	(v: T, k: number) => number	
optional thisArg	any	
Return Value	IOBuffer	

IOBuffer.isBuffer(value)

static

Returns whether a value is an `IOBuffer`.

```
static isBuffer(value: any): boolean;
```

Name	Type	Description
value	any	
Return Value	boolean	

slice(start, end)

Returns a new `IOBuffer` over a byte range (a view, like `Uint8Array.slice`).

```
slice(start?: number, end?: number): IOBuffer;
```

Name	Type	Description
optional start	number	
optional end	number	
Return Value	IOBuffer	

toString(encoding, start, end)

Decodes (a range of) the buffer to a string.

```
toString(encoding?: IOBufferEncoding, start?: number, end?: number):
string;
```

Name	Type	Description
optional encoding	IOBufferEncoding	the output encoding (default `"utf8"`).
optional start	number	start byte offset.
optional end	number	end byte offset (exclusive).
Return Value	string	

toUint8Array()

Returns a plain `Uint8Array` view of the same bytes.

```
toUint8Array(): Uint8Array;
```

Name	Type	Description
Return Value	Uint8Array	

type IOBufferEncoding

Encodings accepted by buffer.IOBuffer construction and conversion.

```
type IOBufferEncoding = "utf8" | "ascii" | "base64" | "hex";
```

class Buffer

Index 40
new Buffer(name) 41
channelcount() 41
create(name, filename, duration, channelcount) 42
framecount() 43
freepeer() 43
getattr(name) 44
getname() 44
length() 45
peek(channel, frame, count) 45
poke(channel, frame, samples) 46
send(message, args) 46
setattr(name, value) 47
setname(name) 47

Bind to a Max `buffer~` object.

The Buffer object in JavaScript is a companion to the buffer~ object in Max. Through it, you can to access samples and metadata for the buffer~ object with the given name.

Index

Constructors

- new Buffer(name)

Methods

- channelcount()
- create(name, filename, duration, channelcount)
- framecount()
- freepeer()
- getattr(name)
- getname()
- length()
- peek(channel, frame, count)
- poke(channel, frame, samples)
- send(message, args)
- setattr(name, value)
- setname(name)

new Buffer(name)

```
new Buffer(name: string);
```

Constructs a new instance of the `Buffer` class

Parameter	Type	Description
name	string	name of the Max `buffer~` to bind to.

channelcount()

Buffer channel count

Return the channels the buffer~ object. This is a method, not a property, and so must be called to get the number of channels.

```
channelcount(): number;
```

Name	Type	Description
Return Value	number	The number of channels in the buffer~ object.

create(name, filename, duration, channelcount)

Supported only in the `v8` *engine.*

Create a new Max `buffer~` object.

```
create(
  name?: string,
  filename?: string,
  duration?: number,
  channelcount?: number,
): void;
```

Name	Type	Description
optional name	string	name of the Max `buffer~` to create. If no name is provided the current name is used.
optional filename	string	filename of the audio file to load. If no filename is provided the buffer is created empty.

optional duration	number	duration of the buffer in milliseconds. If no duration is provided the buffer is created empty.
optional channelcount	number	number of channels in the buffer. If no channelcount is provided the buffer is created with 1 channel.

framecount()

Buffer frame count

Return the number of frames (samples in a single channel) in the buffer~ object. This is a method, not a property, and so must be called to get the number of frames.

```
framecount(): number;
```

Name	Type	Description
Return Value	number	The number of frames in the buffer~ object.

freepeer()

Free the native C peer

Frees the buffer~ data from the native C peer (created when making a Buffer object), which is not considered by the JavaScript garbage collector, and may consume lots of memory until the garbage collector decides to run based JS allocated memory. Once called, the Buffer object is not available for any other use. It's not necessary to call this function, as the memory will be freed eventually, but you can call it whenever you're done with your Buffer object.

```
freepeer(): void;
```

getattr(name)

Supported only in the `v8` *engine.*

Get the value of the named attribute

```
getattr(name: string): number | string | (number | string)[];
```

Name	Type	Description
name	string	the name of the attribute to retrieve
Return Value	number | string | (number | string)[]	the value of the attribute, as an array if the attribute value is a list

getname()

Supported only in the `v8` *engine.*

Get the name of the buffer

```
getname(): string;
```

Name	Type	Description
Return Value	string	The name of the buffer~ object.

length()

Length of the buffer in milliseconds

Return the length of the buffer~ object in milliseconds. This is a method, not a property, and so must be called to get the length of the buffer.

```
length(): number;
```

Name	Type	Description
Return Value	number	The length of the buffer in milliseconds.

peek(channel, frame, count)

Fetch an array of samples from the buffer

```
peek(channel: number, frame: number, count: number): number[];
```

Name	Type	Description
channel	number	channel to fetch samples from (indexed from 1)
frame	number	frame at which to start fetching samples (indexed from 0)

count	number	number of samples to fetch
Return Value	number[]	

poke(channel, frame, samples)

Write samples into the buffer

It is more efficient to call this function once with an array, than to call it mulpitle times, each time with a single sample.

```
poke(channel: number, frame: number, samples: number | number[]): void;
```

Name	Type	Description
channel	number	channel to write samples to (indexed from 1)
frame	number	frame at which to start writing samples (indexed from 0)
samples	number | number[]	samples to write (or a single sample to write)

send(message, args)

Send a message to the buffer.

Can send any message that buffer~ understands. See buffer~ reference.

```
send(message: string, ...args: any[]): void;
```

Name	Type	Description
message	string	the name of message
args	any[]	arguments that follow the name of the message

setattr(name, value)

Supported only in the `v8` *engine.*

Set the value of the named attribute

```
setattr(name: string, value: number | string | (number | string)[]): void;
```

Name	Type	Description
name	string	the name of the attribute to set
value	number | string | (number | string)[]	the value of the attribute to set

setname(name)

Supported only in the `v8` *engine.*

Set the name of the buffer

```
setname(name: string): void;
```

Name	Type	Description
name	string	the name of the buffer to set

interface Console

Index 49
assert(condition, data) 50
count(label) 50
countReset(label) 51
debug(data) 51
error(data) 52
info(data) 52
log(data) 52
time(label) 53
timeEnd(label) 53
timeLog(label, data) 54
trace(data) 54
warn(data) 55

The Max-console-backed `console` object exposed by the v8 runtime. Output is routed to the Max console: `log`, `info`, and `debug` use normal post rows, `warn` uses warning rows, and `error`, `assert`, and `trace` use error rows.

Formatting is a pragmatic subset: string joining, `Error` stack/message rendering, JSON-ish object rendering with a circular fallback, and the `%s`, `%d`, `%i`, `%f`, `%j`, `%o`, `%O`, and `%%` format tokens. Full Node `util.inspect`, `console.table`, grouping, profiling, and configurable output streams are not implemented.

In the legacy `[js]` / `[jsui]` (SpiderMonkey) objects, use post() and error() instead.

Index

Methods

- assert(condition, data)
- count(label)
- countReset(label)
- debug(data)
- error(data)
- info(data)
- log(data)
- time(label)
- timeEnd(label)
- timeLog(label, data)
- trace(data)
- warn(data)

assert(condition, data)

If `condition` is falsy, prints the remaining arguments using Max error rows. If `condition` is truthy, does nothing.

```
assert(condition?: boolean, ...data: any[]): void;
```

Name	Type	Description
optional condition	boolean	the assertion to test
data	any[]	values to format and print when the assertion fails

count(label)

Increments and prints a counter for the given label.

```
count(label?: string): void;
```

Name	Type	Description
optional label	string	the counter label (default `"default"`)

countReset(label)

Resets the counter for the given label.

```
countReset(label?: string): void;
```

Name	Type	Description
optional label	string	the counter label (default `"default"`)

debug(data)

Prints a debug message using normal Max console rows.

```
debug(...data: any[]): void;
```

Name	Type	Description
data	any[]	values to format and print

error(data)

Prints an error using Max error rows.

```
error(...data: any[]): void;
```

Name	Type	Description
data	any[]	values to format and print

info(data)

Prints an informational message. Alias of Console.log() routing to normal Max console rows.

```
info(...data: any[]): void;
```

Name	Type	Description
data	any[]	values to format and print

log(data)

Prints a message using normal Max console rows.

```
log(...data: any[]): void;
```

Name	Type	Description
data	any[]	values to format and print

time(label)

Starts a timer under the given label.

```
time(label?: string): void;
```

Name	Type	Description
optional label	string	the timer label (default `"default"`)

timeEnd(label)

Stops the timer for the given label and prints the elapsed time.

```
timeEnd(label?: string): void;
```

Name	Type	Description
optional label	string	the timer label (default `"default"`)

timeLog(label, data)

Prints the elapsed time for the given label without stopping the timer.

```
timeLog(label?: string, ...data: any[]): void;
```

Name	Type	Description
optional label	string	the timer label (default `"default"`)
data	any[]	additional values to print alongside the elapsed time

trace(data)

Prints a stack trace using Max error rows.

```
trace(...data: any[]): void;
```

Name	Type	Description
data	any[]	optional values to format and print with the trace

warn(data)

Prints a warning using Max warning rows.

```
warn(...data: any[]): void;
```

Name	Type	Description
data	any[]	values to format and print

module dgram

Index 57
function createSocket 57
interface RemoteInfo 58
address 58
port 58
class Socket 58
addMembership(multicastAddress, interfaceAddress) 59
address() 60
bind(port, address, callback) 60
close() 60
dropMembership(multicastAddress, interfaceAddress) 61
off(event, listener) 61
on(event, listener) 62
once(event, listener) 62
send(msg, port, address, callback) 62
setBroadcast(flag) 63
setMulticastInterface(iface) 63
setMulticastLoopback(flag) 64
setMulticastTTL(ttl) 64
setTTL(ttl) 65
type 65

Supported only in the `v8` engine. Introduced in Max v9.2.0.

The `dgram` module exposed by `require("dgram")`.

IPv6 parity is limited and UDP socket-option coverage is narrow.

Example

```
const dgram = require("dgram");
const socket = dgram.createSocket("udp4");
socket.on("message", (msg, rinfo) =>
  post(`udp ${rinfo.address}:${rinfo.port} ${msg.toString()}`, "\n"),
);
socket.bind(41234, "127.0.0.1", () => socket.send("ping", 41234,
"127.0.0.1"));
```

Index

Functions

- createSocket(type)

Classes

- Socket

Interfaces

- RemoteInfo

function createSocket

Creates a UDP dgram.Socket.

```
function createSocket(type: "udp4" | "udp6"): Socket;
```

Name	Type	Description
type	"udp4" | "udp6"	
Return Value	Socket	

interface RemoteInfo

The sender information passed with a UDP `message` event.

Properties

- address
- port

address string

port number

class Socket

A UDP socket.

Incoming `message` payloads arrive as buffer.IOBuffer. Socket-option setters throw a structured error (`syscall: "setsockopt"`) on failure and lazily open the socket if it has not been bound. Events: `message` , `listening` , `close` , `error` .

Properties

- type

Methods

- addMembership(multicastAddress, interfaceAddress)
- address()
- bind(port, address, callback)
- close()
- dropMembership(multicastAddress, interfaceAddress)
- off(event, listener)
- on(event, listener)
- once(event, listener)
- send(msg, port, address, callback)
- setBroadcast(flag)
- setMulticastInterface(iface)
- setMulticastLoopback(flag)
- setMulticastTTL(ttl)
- setTTL(ttl)

addMembership(multicastAddress, interfaceAddress)

Joins a multicast group.

```
addMembership(multicastAddress: string, interfaceAddress?: string): void;
```

Name	Type	Description
multicastAddress	string	
optional interfaceAddress	string	

address()

Returns the bound local endpoint.

```
address(): AddressInfo;
```

Name	Type	Description
Return Value	AddressInfo	

bind(port, address, callback)

Binds the socket to a local port/address.

```
bind(port: number, address?: string, callback?: () => void): void;
```

Name	Type	Description
port	number	
optional address	string	
optional callback	() => void	

close()

Closes the socket.

```
close(): void;
```

dropMembership(multicastAddress, interfaceAddress)

Leaves a multicast group.

```
dropMembership(multicastAddress: string, interfaceAddress?: string): void;
```

Name	Type	Description
multicastAddress	string	
optional interfaceAddress	string	

off(event, listener)

Removes an event listener.

```
off(event: string, listener: (...args: any[]) => void): this;
```

Name	Type	Description
event	string	
listener	(...args: any[]) => void	
Return Value	this	

on(event, listener)

Registers an event listener.

```
on(event: string, listener: (...args: any[]) => void): this;
```

Name	Type	Description
event	string	
listener	(...args: any[]) => void	
Return Value	this	

once(event, listener)

Registers a one-shot event listener.

```
once(event: string, listener: (...args: any[]) => void): this;
```

Name	Type	Description
event	string	
listener	(...args: any[]) => void	
Return Value	this	

send(msg, port, address, callback)

Sends a datagram.

```
send(
  msg: string | buffer.IOBuffer | ArrayBufferView,
  port: number,
  address?: string,
  callback?: (error?: Error) => void,
): void;
```

Name	Type	Description
msg	string | buffer.IOBuffer | ArrayBufferView	
port	number	
optional address	string	
optional callback	(error?: Error) => void	

setBroadcast(flag)

Enables or disables broadcast.

```
setBroadcast(flag: boolean): void;
```

Name	Type	Description
flag	boolean	

setMulticastInterface(iface)

Selects the outgoing multicast interface (IPv4 dotted-quad only).

```
setMulticastInterface(iface: string): void;
```

Name	Type	Description
iface	string	

setMulticastLoopback(flag)

Enables or disables multicast loopback.

```
setMulticastLoopback(flag: boolean): void;
```

Name	Type	Description
flag	boolean	

setMulticastTTL(ttl)

Sets the multicast time-to-live.

```
setMulticastTTL(ttl: number): void;
```

Name	Type	Description
ttl	number	

setTTL(ttl)

Sets the unicast time-to-live.

```
setTTL(ttl: number): void;
```

Name	Type	Description
ttl	number	

type string

read-only

The socket protocol, `"udp4"` or `"udp6"`.

class Dict

Index 67
new Dict(name) 68
append(key, values) 69
clear() 69
clone(name) 69
contains(key) 70
export_json(filename) 70
export_yaml(filename) 71
freepeer() 71
get(key) 72
getkeys() 72
getnames() 72
getsize(key) 73
gettype(key) 73
import_json(filename) 74
import_yaml(filename) 74
name 75
parse(serialization) 75
pull_from_coll(coll_name) 75
push_to_coll(coll_name) 76
quiet 76
readany(filename) 76
remove(key) 77
replace(key, value) 77
set(key, value) 78
setparse(key, serialization) 79
stringify() 79
stringify_compressed() 79
stringify_unformatted() 80
writeagain() 80

Bind to a Max `dict` object.

The Dict object lets you access structured data (a dictionary) associated with a name. If there is a named `dict` object in Max, it will share its contents with the JavaScript Dict object of the same name.

Index

Constructors

- new Dict(name)

Properties

- name
- quiet

Methods

- append(key, values)
- clear()
- clone(name)
- contains(key)
- export_json(filename)
- export_yaml(filename)
- freepeer()
- get(key)
- getkeys()
- getnames()
- getsize(key)
- gettype(key)

- import_json(filename)
- import_yaml(filename)
- parse(serialization)
- pull_from_coll(coll_name)
- push_to_coll(coll_name)
- readany(filename)
- remove(key)
- replace(key, value)
- set(key, value)
- setparse(key, serialization)
- stringify()
- stringify_compressed()
- stringify_unformatted()
- writeagain()

new Dict(name)

```
new Dict(name?: string);
```

Create a named dictionary

If no name is provided as an argument then a unique name will be generated for the dictionary.

Parameter	Type	Description
optional name	string	name of the dictionary

append(key, values)

Add values to the associated data

Add values to the end of an array associated with the specified key.

```
append(key: string, ...values: any[]): void;
```

Name	Type	Description
key	string	the entry key
values	any[]	

clear()

Erase the contents of the dictionary, restoring to a clean state.

```
clear(): void;
```

clone(name)

Copy the named dictionary into this dictionary

```
clone(name: string): void;
```

Name	Type	Description
name	string	name of the dictionary to copy

Example

```
var d1 = new Dict("one");
var d2 = new Dict("two");

d1.clone("two"); // Copies the dictionary named "two" into d1
```

contains(key)

Return a 0 or 1 indicating the specified key exists (or doesn't) in the dictionary.

```
contains(key: string): number;
```

Name	Type	Description
key	string	name of the key to lookup
Return Value	number	

export_json(filename)

Write a file in the JSON format.

```
export_json(filename: string): void;
```

Name	Type	Description
filename	string	Name of the file to read

export_yaml(filename)

Write a file in the YAML format.

```
export_yaml(filename: string): void;
```

Name	Type	Description
filename	string	Name of the file to read

freepeer()

Free the native C peer

Frees the dictionary data from the native C peer (created when making a Dict object), which is not considered by the JavaScript garbage collector, and may consume lots of memory until the garbage collector decides to run based JS allocated memory. Once called, the Dict object is not available for any other use. It's not necessary to call this function, as the memory will be freed eventually, but you can call it whenever you're done with your Dict object.

```
freepeer(): void;
```

get(key)

Return the value associated with a key.

```
get(key: string): any;
```

Name	Type	Description
key	string	lookup key
Return Value	any	

getkeys()

Return a list of all the keys in a dictionary, or null if the dictionary contains no keys.

```
getkeys(): string[] | null;
```

Name	Type	Description
Return Value	string[] | null	

getnames()

List of all named dictionaries.

Return a list of all the dictionary names that are known to Max.

```
getnames(): string[];
```

Name	Type	Description
Return Value	string[]	

getsize(key)

Length of the array of values associated with a key.

```
getsize(key: string): number;
```

Name	Type	Description
key	string	The key to look up
Return Value	number	

gettype(key)

Type of the value or values associated with a key.

```
gettype(key: string): string;
```

Name	Type	Description
key	string	The key to look up
Return Value	string	

import_json(filename)

Read a file in the JSON format.

```
import_json(filename: string): void;
```

Name	Type	Description
filename	string	Name of the file to read

import_yaml(filename)

Read a file in the YAML format.

```
import_yaml(filename: string): void;
```

Name	Type	Description
filename	string	Name of the file to read

name string

The name of the dictionary

parse(serialization)

Parse a serialized dictionary and replace contents

Replaces the contents of the dictionary by parsing a dictionary serialization. Understands JSON and Max Dictionary Syntax

```
parse(serialization: string): void;
```

Name	Type	Description
serialization	string	serialized dictionary string

pull_from_coll(coll_name)

Pull data from a named `coll` object

Adds entries to the dictionary by pulling rows from the given `coll` object. Does not clear existing keys from the dictionary.

```
pull_from_coll(coll_name: string): void;
```

Name	Type	Description
coll_name	string	name of the `coll` object in Max

push_to_coll(coll_name)

Pull data into a named `coll` object

Push the dictionary's content into a named `coll` object. The keys in the dictionary will become the indices in the `coll`, and the values for those indices the values of the dictionary's keys.

```
push_to_coll(coll_name: string): void;
```

Name	Type	Description
coll_name	string	name of the `coll` object in Max

quiet boolean

Suppresses many errors or warnings if set to true

readany(filename)

Loads the contents of a file.

Replaced the contents of the dictionary, clearing existing keys. Accepts JSON and YAML files.

```
readany(filename: string): void;
```

Name	Type	Description
filename	string	name of the file to load

remove(key)

Remove a key and its associated value from the dictionary.

```
remove(key: string): void;
```

Name	Type	Description
key	string	the key to remove

replace(key, value)

Set the value for a key to a specified value, creating a nested dicts if necessary.

Unlike Dict.set(), this function will create a hierarchical path to a given value if one does not already exist.

```
replace(key: string, ...value: any[]): void;
```

Name	Type	Description
key	string	The key or path to a dictionary entry
value	any[]	The value or values to store at that path

Example

```
var d = new Dict(); // empty dictionary
d.replace("first::second::third", 42);
post(d.get("first").get("second").get("third")); // 42
```

set(key, value)

Set the value for a key to a specified value.

Unlike Dict.replace(), will not create nested dictionaryies if the nested structure does not already exist.

```
set(key: string, ...value: any[]): void;
```

Name	Type	Description
key	string	The key or path to a dictionary entry
value	any[]	The value or values to store at that path

setparse(key, serialization)

Set the value for a key using a serialized dictionary

The serialization can be formatted as JSON or as Max Dictionary Syntax

```
setparse(key: string, serialization: string): void;
```

Name	Type	Description
key	string	The key or path to a dictionary entry
serialization	string	Will be parsed to a dictionary and added to the Dict

stringify()

Return the content of the dictionary as a JSON string.

```
stringify(): string;
```

Name	Type	Description
Return Value	string	

stringify_compressed()

Return the content of the dictionary as a compressed JSON string.

```
stringify_compressed(): string;
```

Name	Type	Description
Return Value	string	

stringify_unformatted()

Return the content of the dictionary as an unformatted JSON string.

```
stringify_unformatted(): string;
```

Name	Type	Description
Return Value	string	

writeagain()

Open a save dialog to write the dictionary contents to a file.

```
writeagain(): void;
```

type DrawingPrimitiveType

Primitive type to use for drawing shapes. See Sketch.shapeprim().

```
export declare type DrawingPrimitiveType =
  | "lines"
  | "line_loop"
  | "line_strip"
  | "points"
  | "polygon"
  | "quads"
  | "quad_grid"
  | "quad_strip"
  | "triangles"
  | "tri_grid"
  | "tri_fan"
  | "tri_strip";
```

class EventSource

Index 83
new EventSource(url, options) 84
addEventListener(type, listener) 84
close() 85
CLOSED 85
CONNECTING 85
onerror 85
onmessage 85
onopen 86
OPEN 86
readyState 86
removeEventListener(type, listener) 86
url 87
withCredentials 87

Supported only in the `v8` *engine. Introduced in Max v9.2.0.*

A Server-Sent Events client.

URLs must be absolute `http://` or `https://` URLs. Successful connections require `200 OK` and a `content-type` beginning with `text/event-stream`. On failure the client dispatches `error` and reconnects (default delay 3000 ms, overridable by the stream's `retry:` field).

Example

```
const source = new EventSource("http://127.0.0.1:8093/sse/basic");
source.onopen = () => post("sse open\n");
source.onmessage = (event) => {
  post(event.data, "\n");
  source.close();
};
source.onerror = () => post("sse error\n");
```

Index

Constructors

- new EventSource(url, options)

Properties

- CLOSED
- CONNECTING
- onerror
- onmessage
- onopen
- OPEN
- readyState
- url
- withCredentials

Methods

- addEventListener(type, listener)

- close()
- removeEventListener(type, listener)

new EventSource(url, options)

```
new EventSource(url: string, options?: { withCredentials?: boolean });
```

Constructs a new instance of the `EventSource` class

Parameter	Type	Description
url	string	an absolute `http://` or `https://` URL.
optional options	object	`withCredentials` controls cross-origin cookie use. *optional* options.withCredentials boolean

addEventListener(type, listener)

Registers a listener. Use a custom `type` to receive named `event:` messages from the stream.

```
addEventListener(
  type: string,
  listener: (event: EventSourceMessageEvent) => void,
): void;
```

Name	Type	Description
type	string	

listener	(event: EventSourceMessageEvent) => void

close()

Closes the connection permanently (no reconnect).

```
close(): void;
```

CLOSED number read-only

Closed state constant.

CONNECTING number read-only

Connecting state constant.

onerror ((event: { type: string }) => void) | null

Called on error (before an automatic reconnect).

onmessage ((event: EventSourceMessageEvent) => void) | null

Called when a default (unnamed) message arrives.

onopen ((event: { type: string }) => void) | null

Called when the connection opens.

OPEN number read-only

Open state constant.

readyState number read-only

The connection state.

removeEventListener(type, listener)

Removes an event listener.

```
removeEventListener(
  type: string,
  listener: (event: EventSourceMessageEvent) => void,
): void;
```

Name	Type	Description
type	string	
listener	(event: EventSourceMessageEvent) => void	

url string read-only

The stream URL.

withCredentials boolean read-only

Whether cross-origin credentials are used.

interface EventSourceMessageEvent

Index 88
data 88
lastEventId 88
origin 89
target 89
type 89

An event delivered to an EventSource.

Index

Properties

- data
- lastEventId
- origin
- target
- type

data string

The event payload.

lastEventId string

The last event ID seen, used as `Last-Event-ID` on reconnect.

origin string

The event origin.

target EventSource

type string

class File

Index 91
new File(filename, access, typelist) 92
access 93
byteorder 94
close(text) 94
eof 94
filename 94
filetype 95
foldername 95
isopen 95
linebreak 95
open(filename) 95
position 96
readbytes(count) 96
readchars(count) 96
readfloat32(count) 97
readfloat64(count) 97
readint16(count) 98
readint32(count) 99
readline(maximumCount) 99
readstring(count) 100
typelist 100
writebytes(bytes) 100
writechars(chars) 101
writefloat32(floats) 101
writefloat64(floats) 102
writeint16(ints) 102
writeint32(ints) 103
writeline(text) 103
writestring(text) 104

The File object provides a means of reading and writing files from JavaScript.

Index

Constructors

- new File(filename, access, typelist)

Properties

- access
- byteorder
- eof
- filename
- filetype
- foldername
- isopen
- linebreak
- position
- typelist

Methods

- close(text)
- open(filename)
- readbytes(count)
- readchars(count)
- readfloat32(count)
- readfloat64(count)
- readint16(count)

- readint32(count)
- readline(maximumCount)
- readstring(count)
- writebytes(bytes)
- writechars(chars)
- writefloat32(floats)
- writefloat64(floats)
- writeint16(ints)
- writeint32(ints)
- writeline(text)
- writestring(text)

new File(filename, access, typelist)

```
new File(
  filename?: string,
  access?: FileAccess,
  typelist?: FileFourCharacterCode[],
);
```

Create a file reference for reading or writing

By default, `typelist` is empty. If `filename` includes an extension, then it is not necessary to supply a `typelist`. If `filename` does not include an extension, then File will look for a file with one of the extensions specified by `typelist`. Note that the four-character code may not match the file extension.

Given arguments, the `File` constructor will open the file automatically if the file can be opened.

Parameter	Type	Description
optional filename	string	File to open. Can be relative, absolute, or anything in the Max search path
optional access	FileAccess	Access mode. Can be "read", "write", or "readwrite"
optional typelist	FileFourCharacterCode[]	Any of the four character codes in max-fileformats.txt

Examples

```
var f = new File("myfile.txt", "write");
post(f.isopen); // true, if myfile.txt is in the Max search path
f.close();
```

It's also possible to create an empty `File` object, and to configure it after the fact. In this case, the `File` object must be opened explicitly.

```
var f = new File();
f.filename = "myfile.txt";
f.access = "write";
f.open();
post(f.isopen); // true, if myfile.txt is in the Max search path
f.close();
```

If for some reason the file could not be opened, then File.isopen will return false.

access FileAccess

File access permissions: "read", "write", or "readwrite". By default, this value is "read".

byteorder FileEndianness

The assumed file byteorder (endianness): "big", "little", or "native". By default, this value is "native".

close(text)

Closes the currently open file.

```
close(text?: string): void;
```

Name	Type	Description
optional text	string	unclear?

eof number

The location of the end of file, in bytes. If set past the end of current file, `File` will append NULL bytes.

filename string

The current filename.

filetype FileFourCharacterCode

The four-character code for the file type.

foldername string read-only

The absolute path to parent folder.

isopen boolean read-only

Return a true/false indicating if the File constructor is successful in finding and opening the file.

linebreak FileLineEndingStyle

The line break convention to use when writing lines. Defaults to "native"

open(filename)

Opens the file specified by the filename argument. If no argument is specified, it will open the last opened file, or the value stored in the `filename` property. Check File.isopen to see if the file was openend successfully.

```
open(filename?: string): void;
```

Name	Type	Description
optional filename	string	the file to open

position number

The current file position, in bytes. Set this to offset the file write position forward or backwards.

readbytes(count)

Read bytes to an array

Reads and returns an array containing up to `count` numbers, read as bytes from the file, starting at the current file position. The file position is updated accordingly.

```
readbytes(count: number): number[];
```

Name	Type	Description
count	number	maximum number of bytes to read
Return Value	number[]	

readchars(count)

Read bytes to an array of single character strings

Reads and returns an array containing the single character strings, read as characters from the file, starting at the current file position. The file position is updated accordingly.

```
readchars(count: number): string[];
```

Name	Type	Description
count	number	maximum number of bytes to read
Return Value	string[]	

readfloat32(count)

Read from a file as 32-bit floating point numbers

Reads and returns an array containing the numbers read as 32-bit floating point numbers from the file, starting at the current file position. The byteorder property is taken into account when reading these values. The file position is updated accordingly.

```
readfloat32(count: number): number[];
```

Name	Type	Description
count	number	maximum number of floats to read
Return Value	number[]	

readfloat64(count)

Read from a file as 64-bit floating point numbers

Reads and returns an array containing the numbers read as 64-bit floating point numbers from the file, starting at the current file position. The byteorder property is taken into account when reading these values. The file position is updated accordingly.

```
readfloat64(count: number): number[];
```

Name	Type	Description
count	number	maximum number of floats to read
Return Value	number[]	

readint16(count)

Read from a file as 16-bit signed integers

Reads and returns an array containing the numbers read as signed 16-bit integers from the file, starting at the current file position. The byteorder property is taken into account when reading these values. The file position is updated accordingly.

```
readint16(count: number): number[];
```

Name	Type	Description
count	number	maximum number of integers to read
Return Value	number[]	

readint32(count)

Read from a file as 32-bit signed integers

Reads and returns an array containing the numbers read as signed 32-bit integers from the file, starting at the current file position. The byteorder property is taken into account when reading these values. The file position is updated accordingly.

```
readint32(count: number): number[];
```

Name	Type	Description
count	number	maximum number of integers to read
Return Value	number[]	

readline(maximumCount)

Read a line of text

Reads and returns a string containing up to `maximumCount` characters or up to the first line break as read from the file, starting at the current file position. The file position is updated accordingly The default maximum count value is 512. This can be increased by specifying a new maximum count as the argument.

```
readline(maximumCount: number): string;
```

Name	Type	Description
maximumCount	number	maximum string length to read
Return Value	string	

readstring(count)

Read a string of text

Reads and returns a string containing up to char_count characters as read from the file, starting at the current file position. Unlike File.readline(), line breaks are not considered. The file position is updated accordingly.

```
readstring(count: number): string;
```

Name	Type	Description
count	number	maximum number of bytes to read
Return Value	string	

typelist FileFourCharacterCode[]

An array file type codes to filter by when opening a file. By default, this is the empty array.

writebytes(bytes)

Write an array of bytes

Writes the numbers contained in the `bytes` argument as bytes to the file, starting at the current file position. The file position is updated accordingly.

```
writebytes(bytes: number[]): void;
```

Name	Type	Description
bytes	number[]	bytes to write

writechars(chars)

Write single character strings

Writes the single character strings contained in the `chars` argument as characters to the file, starting at the current file position. The file position is updated accordingly.

```
writechars(chars: string[]): void;
```

Name	Type	Description
chars	string[]	array of single character strings

writefloat32(floats)

Write numbers as 32-bit floating point numbers

Writes the numbers contained in the `floats` argument as 32-bit floating point numbers to the file, starting at the current file position. The byteorder property is taken into account when writing these values. The file position is updated accordingly.

```
writefloat32(floats: number[]): void;
```

Name	Type	Description
floats	number[]	array of numbers to write as float32

writefloat64(floats)

Write numbers as 64-bit floating point numbers

Writes the numbers contained in the `floats` argument as 64-bit floating point numbers to the file, starting at the current file position. The byteorder property is taken into account when writing these values. The file position is updated accordingly.

```
writefloat64(floats: number[]): void;
```

Name	Type	Description
floats	number[]	array of numbers to write as float64

writeint16(ints)

Write numbers as 16-bit signed integers

Writes the numbers contained in the `ints` argument as signed 16-bit integers to the file, starting at the current file position. The byteorder property is taken into account when writing these values. The file position is updated accordingly.

```
writeint16(ints: number[]): void;
```

Name	Type	Description
ints	number[]	array of numbers to write as int16

writeint32(ints)

Write numbers as 32-bit signed integers

Writes the numbers contained in the `ints` argument as signed 32-bit integers to the file, starting at the current file position. The byteorder property is taken into account when writing these values. The file position is updated accordingly.

```
writeint32(ints: number[]): void;
```

Name	Type	Description
ints	number[]	array of numbers to write as int32

writeline(text)

Write a line of text

Writes the characters contained in the string argument as characters to the file, starting at the current file position, and inserts a line break appropriate to the `linebreak` property. The file position is updated accordingly.

```
writeline(text: string): void;
```

Name	Type	Description
text	string	the text to write

writestring(text)

Write a string of text

Writes the characters contained in the string argument as characters to the file, starting at the current file position. Unlike File.writeline(), no line break is inserted. The file position is updated accordingly.

```
writestring(text: string): void;
```

Name	Type	Description
text	string	the text to write

type FileAccess

File access type Used with File

```
export declare type FileAccess = "read" | "write" | "readwrite";
```

type FileEndianness

File endianness Used with File

```
export declare type FileEndianness = "big" | "little" | "native";
```

type FileFourCharacterCode

Four character file codes. Used with File

```
export declare type FileFourCharacterCode =
  | "iLaF"
  | "maxb"
  | "TEXT"
  | "mx@c"
  | "GenX"
  | "mRef"
  | "cafe"
  | "jar "
  | "WAVE"
  | "wv64"
  | "AIFF"
  | "NxTS"
  | "ULAW"
  | "FLAC"
  | "DATA"
  | "Midi"
  | "JiT!"
  | "maxc"
  | "PICT"
  | "PICS"
  | "MPEG"
  | "mpg4"
  | "MooV"
  | "WVC1"
  | "WMVA"
  | "WMV3"
  | "WMV2"
  | "M4V "
  | "GIFf"
  | "JPEG"
  | "PNG "
  | "PNGf"
  | "TIFF"
  | "SWFL"
  | "8BPS"
  | "BMP "
  | "exr "
  | "VfW "
  | "AFxP"
  | "AFxB"
  | "AUps"
  | "V3ps"
```

```
    =
      | "iLaF"
      | "maxb"
      | "TEXT"
      | "mx@c"
      | "GenX"
      | "mRef"
      | "cafe"
      | "jar "
      | "WAVE"
      | "wv64"
      | "AIFF"
      | "NxTS"
      | "ULAW"
      | "FLAC"
      | "DATA"
      | "Midi"
      | "JiT!"
      | "maxc"
      | "PICT"
      | "PICS"
      | "MPEG"
      | "mpg4"
      | "MooV"
      | "WVC1"
      | "WMVA"
      | "WMV3"
      | "WMV2"
      | "M4V "
      | "GIFf"
      | "JPEG"
      | "PNG "
      | "PNGf"
      | "TIFF"
      | "SWFL"
      | "8BPS"
      | "BMP "
      | "exr "
      | "VfW "
      | "AFxP"
      | "AFxB"
      | "AUps"
```

```
  "iLaF"
    | "maxb"
    | "TEXT"
    | "mx@c"
    | "GenX"
    | "mRef"
    | "cafe"
    | "jar "
    | "WAVE"
    | "wv64"
    | "AIFF"
    | "NxTS"
    | "ULAW"
    | "FLAC"
    | "DATA"
    | "Midi"
    | "JiT!"
    | "maxc"
    | "PICT"
    | "PICS"
    | "MPEG"
    | "mpg4"
    | "MooV"
    | "WVC1"
    | "WMVA"
    | "WMV3"
    | "WMV2"
    | "M4V "
    | "GIFf"
    | "JPEG"
    | "PNG "
    | "PNGf"
    | "TIFF"
    | "SWFL"
    | "8BPS"
    | "BMP "
    | "exr "
    | "VfW "
    | "AFxP"
    | "AFxB"
    | "AUps"
    | "V3ps"
```