Streams API

> **Note:** This feature is available in Web Workers.

The Streams API allows JavaScript to programmatically access streams of data received over the network and process them as desired by the developer.

Concepts and usage

Streaming involves breaking a resource that you want to receive over a network down into small chunks, then processing it bit by bit. Browsers already do this when receiving media assets — videos buffer and play as more of the content downloads, and sometimes you'll see images display gradually as more is loaded too.

But this capability has never been available to JavaScript before. Previously, if we wanted to process a resource of some kind (video, text file, etc.), we'd have to download the entire file, wait for it to be deserialized into a suitable format, then process all the data.

With the Streams API, you can start processing raw data with JavaScript bit by bit, as soon as it is available, without needing to generate a buffer, string, or blob.

There are more advantages too — you can detect when streams start or end, chain streams together, handle errors and cancel streams as required, and react to the speed at which the stream is being read.

The usage of Streams hinges on making responses available as streams. For example, the response body returned by a successful fetch request is a `ReadableStream` that can be read by a reader created with `ReadableStream.getReader()`.

More complicated uses involve creating your own stream using the `ReadableStream()` constructor, for example to process data inside a service worker.

You can also write data to streams using `WritableStream`.

> **Note:** You can find a lot more details about the theory and practice of streams in our articles — Streams API concepts, Using readable streams, Using readable byte streams, and Using writable streams.

Stream interfaces

Readable streams

`ReadableStream`

Represents a readable stream of data. It can be used to handle response streams of the Fetch API, or developer-defined streams (e.g. a custom `ReadableStream()` constructor).

`ReadableStreamDefaultReader`

Represents a default reader that can be used to read stream data supplied from a network (e.g. a fetch request).

`ReadableStreamDefaultController`

Represents a controller allowing control of a `ReadableStream` 's state and internal queue. Default controllers are for streams that are not byte streams.

Writable streams

`WritableStream`

Provides a standard abstraction for writing streaming data to a destination, known as a sink. This object comes with built-in backpressure and queuing.

`WritableStreamDefaultWriter`

Represents a default writable stream writer that can be used to write chunks of data to a writable stream.

`WritableStreamDefaultController`

Represents a controller allowing control of a `WritableStream` 's state. When constructing a `WritableStream` , the underlying sink is given a corresponding `WritableStreamDefaultController` instance to manipulate.

Transform Streams

`TransformStream`

Represents an abstraction for a stream object that transforms data as it passes through a pipe chain of stream objects.

`TransformStreamDefaultController`

Provides methods to manipulate the `ReadableStream` and `WritableStream`

associated with a transform stream.

Related stream APIs and operations

`ByteLengthQueuingStrategy`

Provides a built-in byte length queuing strategy that can be used when constructing streams.

`CountQueuingStrategy`

Provides a built-in chunk counting queuing strategy that can be used when constructing streams.

Extensions to other APIs

`Request`

When a new `Request` object is constructed, you can pass it a `ReadableStream` in the `body` property of its `RequestInit` dictionary. This `Request` could then be passed to a `fetch()` to commence fetching the stream.

`Response.body`

The response body returned by a successful fetch request is exposed by default as a `ReadableStream`, and can have a reader attached to it, etc.

ByteStream-related interfaces

`ReadableStreamBYOBReader`

Represents a BYOB ("bring your own buffer") reader that can be used to read stream data supplied by the developer (e.g. a custom `ReadableStream()` constructor).

`ReadableByteStreamController`

Represents a controller allowing control of a `ReadableStream` 's state and internal queue. Byte stream controllers are for byte streams.

`ReadableStreamBYOBRequest`

Represents a pull into request in a `ReadableByteStreamController` .

Examples

We have created a directory of examples to go along with the Streams API documentation — see mdn/dom-examples/streams . The examples are as follows:

- Simple stream pump : This example shows how to consume a ReadableStream and pass its data to another.
- Grayscale a PNG : This example shows how a ReadableStream of a PNG can be turned into grayscale.
- Simple random stream : This example shows how to use a custom stream to generate random strings, enqueue them as chunks, and then read them back out again.
- Simple tee example : This example extends the Simple random stream example, showing how a stream can be teed and both resulting streams can be read independently.
- Simple writer : This example shows how to write to a writable stream, then decode the stream and write the contents to the UI.
- Unpack chunks of a PNG : This example shows how `pipeThrough()` can be used to transform a ReadableStream into a stream of other data types by transforming a data of a PNG file into a stream of PNG chunks.

Examples from other developers:

- Progress Indicators with Streams, Service Workers, & Fetch .

Specifications

Specification
Streams Standard # rs-class
Streams Standard # ws-class

Browser compatibility

api.ReadableStream

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android
`ReadableStream`	43	14	65	30	10.1	43	65	30
[Symbol.asyncIterator]	124	124	110	110	No	124	110	82
`ReadableStream()` constructor	52	79	65	39	10.1	52	65	41
`cancel`	43	14	65	30	10.1	43	65	30
`from()` static method	No	No	117	No	No	No	117	No

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android
`getReader`	43	14	65	30	10.1	43	65	30
`locked`	52	14	65	39	10.1	52	65	41
`pipeThrough`	59	79	102	46	10.1	59	102	43
`pipeTo`	59	79	100	46	10.1	59	100	43
`tee`	52	79	65	39	10.1	52	65	41
`transferable`	87	87	103	73	No	87	103	62
`values`	124	124	110	110	No	124	110	82

Tip: you can click/tap on a cell for more information.

Full support Partial support No support

Experimental. Expect behavior to change in the future. See implementation notes.

Has more compatibility info.

api.WritableStream

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	
`WritableStream`	59	16	100	47	14.1	59	100	44	14.5	
`WritableStream()` constructor	59	16	100	47	14.1	59	100	44	14.5	
`abort`	59	16	100	47	14.1	59	100	44	14.5	
`close`	81	81	100	68	14.1	81	100	58	14.5	
`getWriter`	59	16	100	47	14.1	59	100	44	14.5	
`locked`	59	16	100	47	14.1	59	100	44	14.5	
`transferable`	87	87	103	73	No	87	103	62	No	

Tip: you can click/tap on a cell for more information.

Full support | Partial support | No support | Has more compatibility info.

See also

- Streams API concepts
- Using readable streams
- Using readable byte streams
- Using writable streams

Yes No

Learn how to contribute.

This page was last modified on Jul 26, 2024 by MDN contributors.