# Stream API - Basics

Stream API allows developers to access the streams of data received over the network and process them bit-by-bit according to their needs with the help of JavaScript programming language. Where the streams are the sequence of data that we want to receive over the network in small chunks and then we process that small chunks in bit-by-bit.

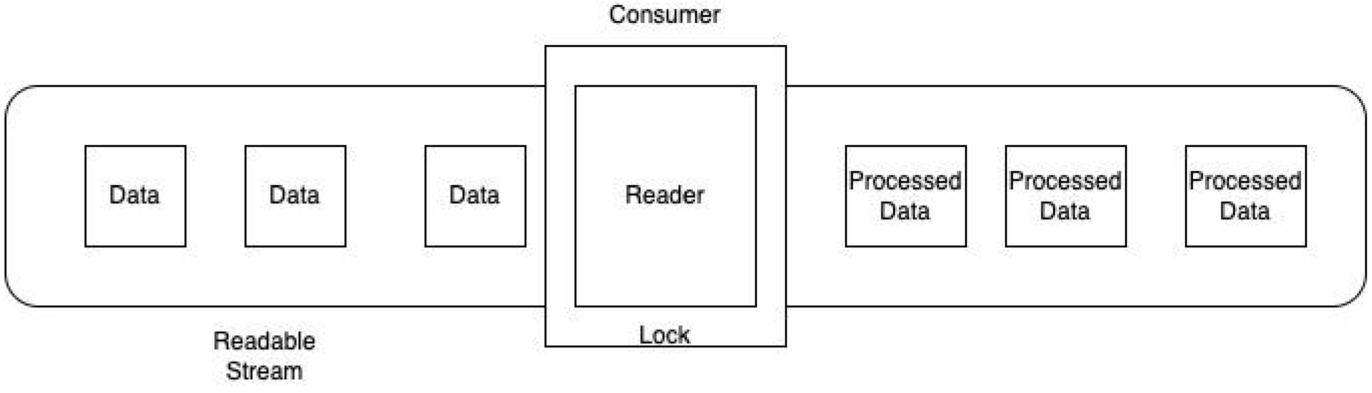
Before streaming, if we want to process a video, audio, or text file we need to download that complete file from the network and wait for it to be deserialized into a specified format and then process the complete downloaded file.

After the introduction of streams, the whole working culture is changed now we can start processing data bit-by-bit as soon as the data is received on the client side using JavaScript without creating any extra buffer or blob. Using streams we can perform various tasks like can find the start and end of the stream or can chain streams together or can easily handle errors, can cancel the streams, and much more.

Streaming is used to create real-world applications like video streaming applications like Netflix, Amazon Prime Videos, Zee5, Voot, YouTube, etc. So that users can easily watch movies, TV shows, etc. online without downloading them. Stream API provides various features like ReadableStream, Teeing, WritableStream, Pipe Chains, Backpressure, internal queue, and queueing strategies. Let's discuss them one by one in detail.

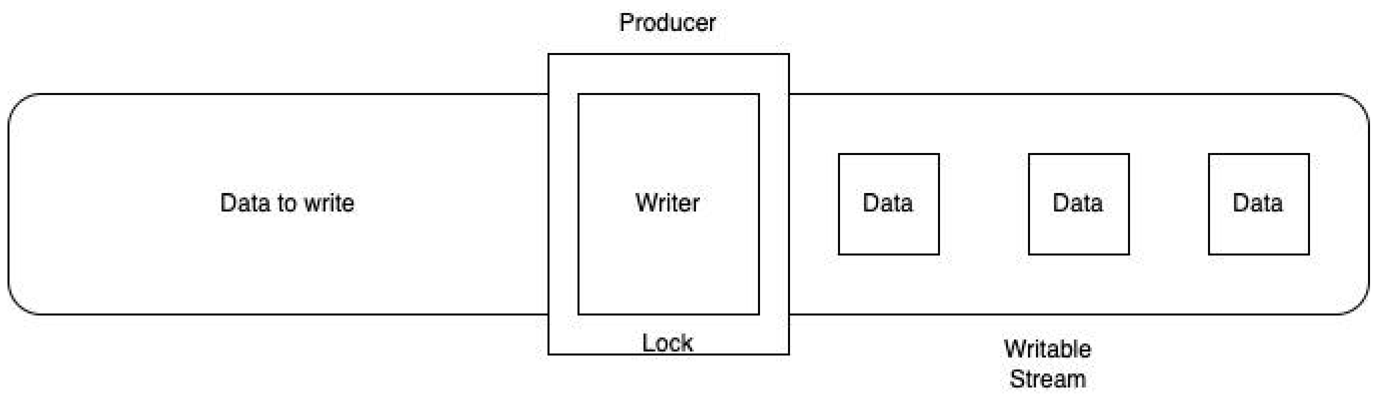
## Readable Stream

The readable stream allows you to read data/chunks from the source in JavaScript using the ReadableStream object. The chunks are small pieces of data that are going to be read by the reader sequentially. It can be of a single bit or can be of a large size like a typed array, etc. To read a readable stream API provide a reader. It reads chunks from the stream and then processes the data of the chunk. Only one reader can read a stream at a time, no other reader is allowed to read that stream.



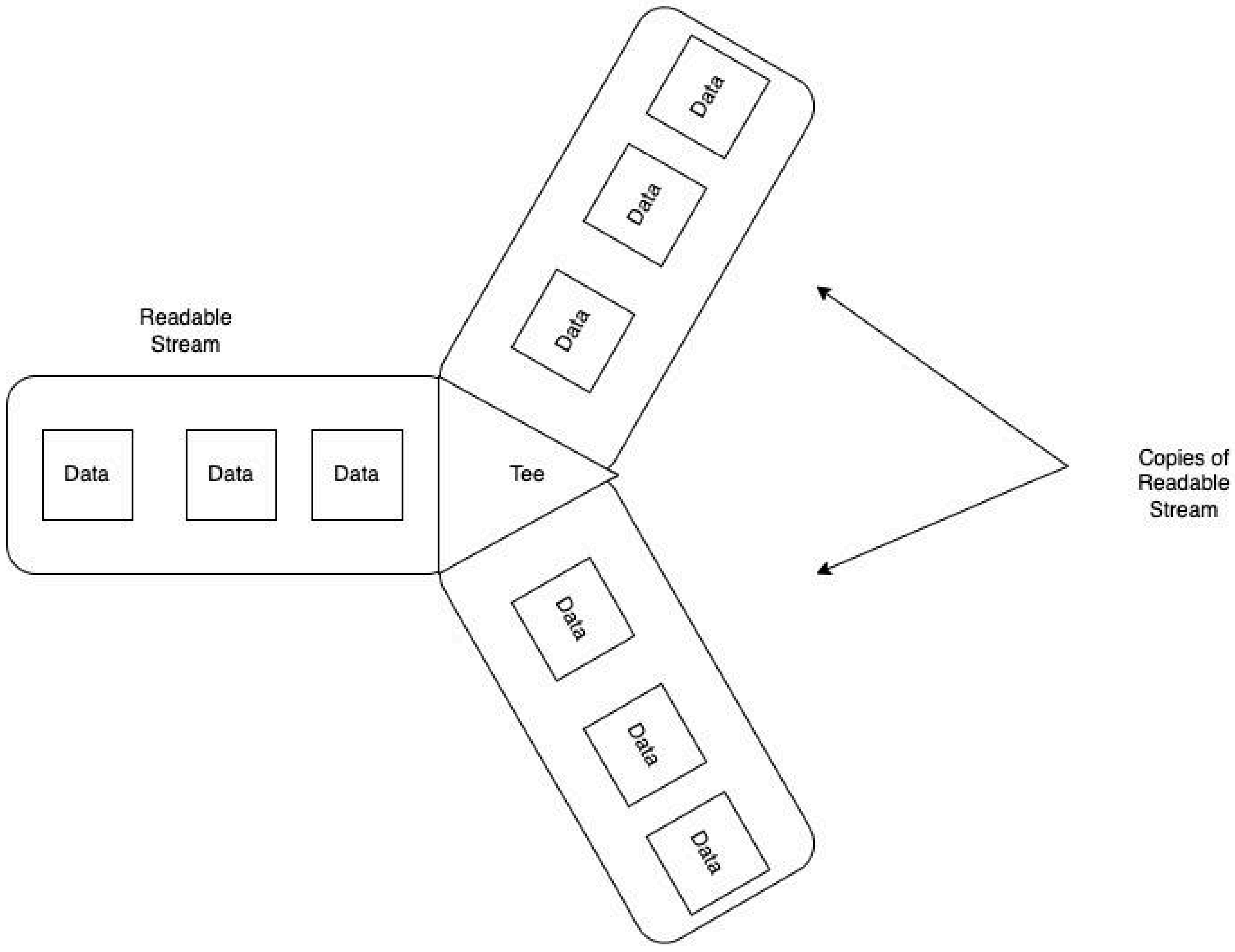
## Writable Stream

The writable stream allows you to write data in JavaScript using the Writable Stream object. The data is written by the writer to the stream. The writer writes data in the form of chunks(one chunk at a time). When the writer is created and starts writing to the stream for that time the stream is locked for that writer, and no other writer is allowed to access that stream and an inter queue is used to keep track of the chunks written by the writer.



## Teeing

Teeing is a process in which a stream is split into two identical copies of the stream so that two separate readers can read the stream at the same time. We can achieve teeing with the help of the ReableStream.tee() method. This method returns an array that contains two identical copies of the specified stream and can be read by two readers.



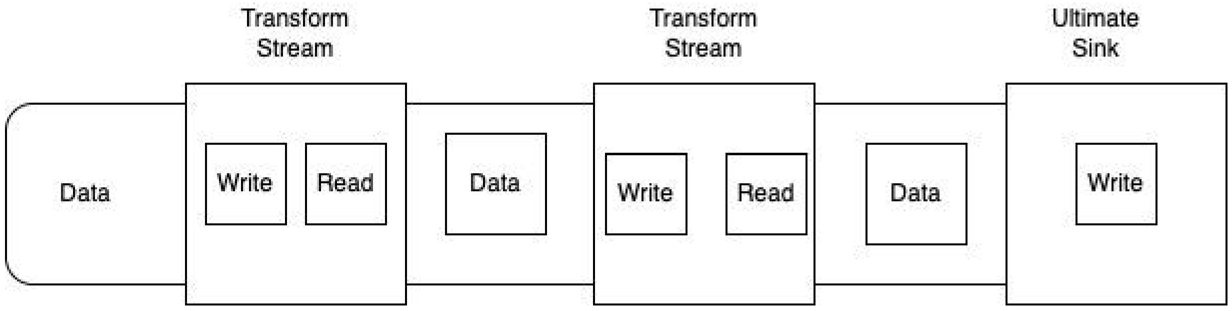
## Pipe Chains

Pipe Chain is a process in which multiple streams are connected together to create a data processing flow. In Stream API, we can pipe one stream into another with the help of a pipe chain structure. The starting point of the pipe chain is known as the original source and the last point of the pipe chain is known as the ultimate sink.

To pipe streams we can use the following methods −

**ReadableStream.pipeThrough()** − This method is used to pipe the current stream through a transform stream. Transform stream contains a pair of readable and writable streams.

**ReadableStream.pipeTo()** − This method is used to pipe the current ReadableStream to the specified WritableStream and will return a promise which resolves when the piping process is successfully completed or rejected due to some error.



## Backpressure

Backpressure is a special concept in Stream API. In this process, a single stream or pipe chain controls the speed of read/write. Suppose we have a stream, this stream is busy and does not able to take new chunks of data, so it sends a backward message through the chain to tell the transform stream to slow down the delivery of chunks so that we can save from bottleneck.

We can use backpressure in the ReadableStream, so we need to find the chunk size required by the consumer with the help of the ReadableStreamDefaultContriller.desiredSize property. If the chunk size is very low, then the ReadableStream can indicate it's an underlying source so stop sending more data and send backpressure along with the stream chain.

When the consumer again wants the received data then we use the pull method to tell the underlying source to send data to the stream.

## Internal queues and queuing Strategies

Internal queues are the queues that keep track of those chunks that have not been processed or finished. For example, in a readable stream, the internal queue keeps track of those chunks that are present on the enqueue but not read yet. Internal queues use a queueing strategy representing how the backpressure signal sends according to the internal queue state.

## Conclusion

So these are the basic concept of Stream API. It is generally used in online streaming. When you watch a video online the browser or the application receives continuous streams of data chunks in the background and then they are processed by that browser or the application to display the video. Now in the next article, we will learn about the Readable stream of Stream API.