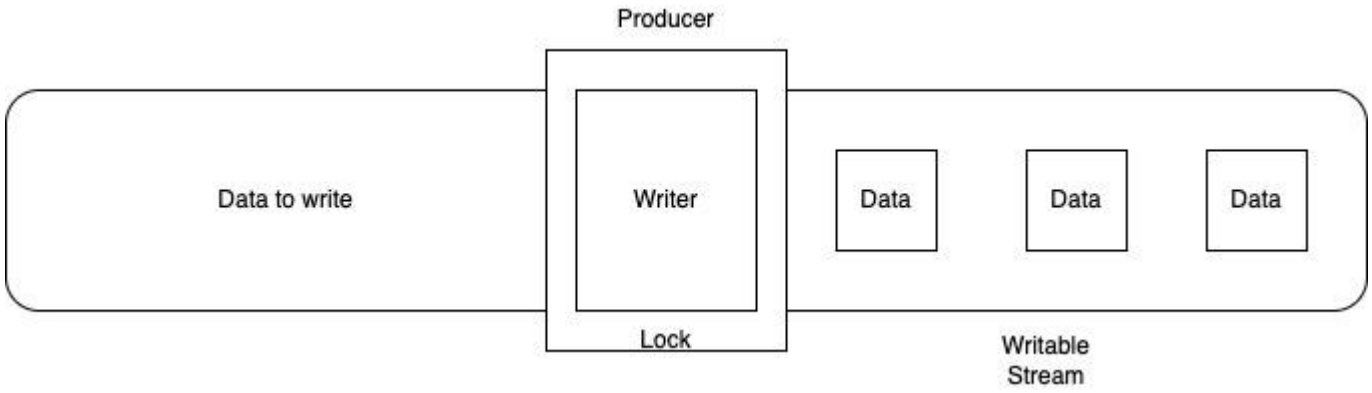
Stream API - Writeable Streams

Writable Streams are those streams in which we can write data. They are generally represented in JavaScript by WritableStrem object. It creates an abstraction over the underlying sink. The underlying sink is a lower-level input/output sink where the raw data is written.

In the writable stream, a writer writes the data. It writes one chunk at a time, where a chunk is a piece of data. Also, you are allowed to use any code to produce the chunk for writing, and the writer and the related code together are known as the producer. On a single stream, only one writer is allowed to write data. At that time the stream is locked for that specified writer, no other writer is allowed to write. If you want another writer to write, then you have to terminate the first writer after that the other writer is allowed to write. Every writer has their own controller which controls the stream.

Also, the writable stream has an internal queue just like a readable stream. It also keeps track of chunks that are written but not processed by the underlying sink.



WritableStream Interfaces

Stream API supports three types of writable stream interfaces –

- WritableStream Interface
- WritableStreamDefaultWriter Interface
- WritableStreamDefaultController Interface

WritableStream Interface

The WritableStream Interface is used to write streaming data to the sink. Its object comes with in-built backpressure and queuing.

Constructor

To create a WritableStream object, the WritableStream interface provides a WritableStream() constructor.

Syntax

```
const newWrite = new WritableStream(UnderlyingSink)
Or
const newWrite = new WritableStream(UnderlyingSink, QueuingStrategy)
```

The WritableStream() constructor has the following optional parameters –

UnderlyingSink – This object provides various methods and properties that show the behaviour of the write stream instance. It takes four parameters: start(controller), write(chunk, controller), close(controller), and abort(reason).

QueuingStrategy – This object is used to define the queuing strategy for the write streams. It takes two parameters: highWaterMark, and size(chunk).

Instance Properties

The properties provided by the WritableStream interface are read-only properties. So the properties provided by WritableStream are –

Sr.No.	Property & Description
1	**WritableStream.locked** This property is used to check whether the WritableStream is locked to the writer or not.

Methods

The following are the commonly used method of the WritableStream interface –

Sr.No.	Property & Description
1	**WritableStream.close()**

	This method is used to close the stream.
2	**WritableStream.abort()** This method is used to abort the stream.
3	**WritableStream.getWriter()** This method is used to get a new object of WriteableStreamDefaultWriter and will lock the stream to that instance. When the stream is locked no other writer can able to get until the current object is released.

WritableStreamDefaultWriter Interface

The WritableStreamDefaultWriter interface is used to represent a default writer which will write chunks of data to the stream.

Constructor

To create a WritableStreamDefaultWriter object, the WritableStreamDefaultWriter interface provides a WritableStreamDefaultWriter() constructor.

Syntax

```
const newWrite = new WritableStreamDefaultWriter(myStream)
```

This constructor contains only one parameter which is myStream. It will read ReadableStream.

Instance Properties

The properties provided by the WritableStreamDefaultWriter interface are read-only properties. So the properties provided by WritableStreamDefaultWriter are –

Sr.No.	Property & Description
1	**WritableStreamDefaultWriter.closed** This property returns a promise which will resolve if the stream is closed or rejected due to some error. It allows you to create a program which will respond at the end of the stream process.
2	**WritableStreamDefaultWriter.desiredSize** This property is used to get the desired size that will fulfil the stream internal queue.

3	**WritableStreamDefaultWriter.ready** This property returns a promise which will resolve when the desired size of the stream internal queue transition from negative to positive.

Methods

The following are the commonly used method of the WritableStreamDefaultWriter interface –

Sr.No.	Method & Description
1	**WritableStreamDefaultWriter.abort()** This method is used to abort the stream.
2	**WritableStreamDefaultWriter.close()** This method is used to close the writable stream.
3	**WritableStreamDefaultWriter.releaseLock()** This method is used to remove the lock of the writer on the corresponding stream.
4	**WritableStreamDefaultWriter.write()** This method is used to write a passed piece of data to a WritableStream and its underlying sink. It will return a promise which resolves to determine whether the write operation is a failure or success.

WritableStreamDefaultController Interface

The WritableStreamDefaultController interface represents a controller which allows us to control the WritableStream State. It does not provide any controller and the instance is created automatically while constructing WritableStream.

Instance Properties

The properties provided by the WritableStreamDefaultController interface are read-only properties. So the properties provided by WritableStreamDefaultController are –

Sr.No.	Property & Description
1	**WritableStreamDefaultController.signal** This property will return an AbortSignal related to the specified controller.

Methods

The following are the commonly used method of the WritableStreamDefaultController interface –

Sr.No.	Method & Description
1	**WritableStreamDefaultController.error()** This method will cause any future interaction with the related write stream to the error.

Example - Creating Writable Stream

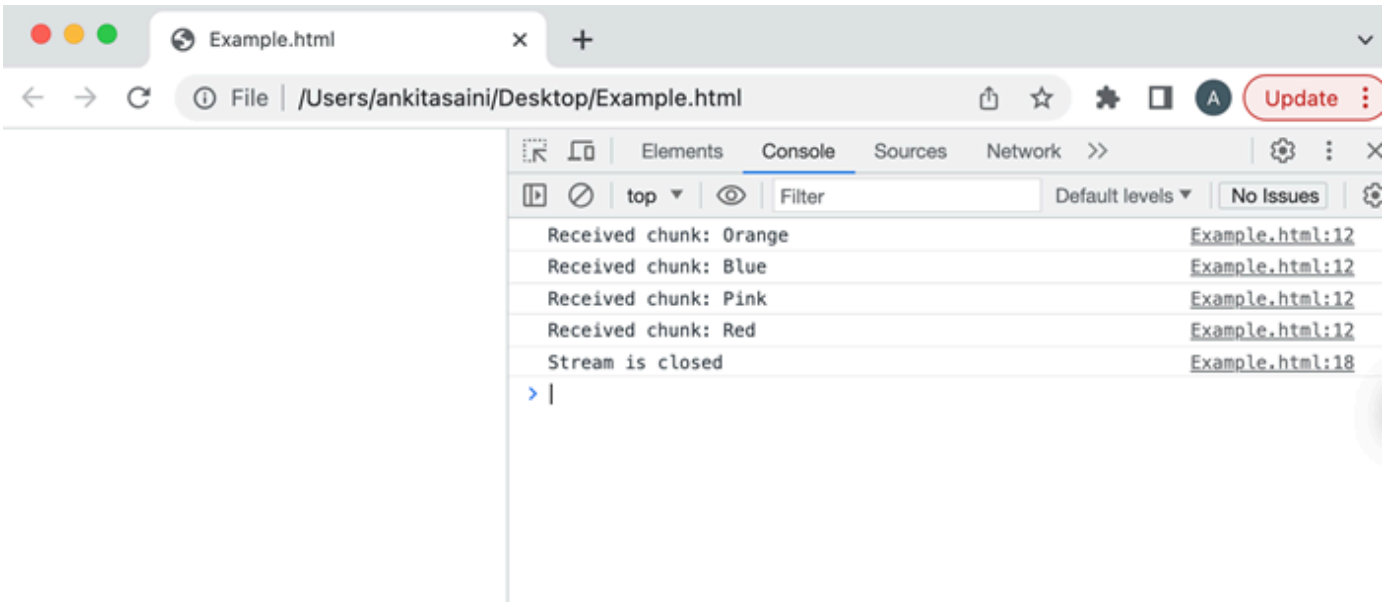
In the following program, we create a custom writable stream. So to create a writable stream Stream API provide WritableStream() constructor with the write(), cancel() and abort() functions. The write() function is used to log the received chunks, the cancel() function is used to handle when a stream is cancelled, and the abort() function is used to handle when the stream is aborted. Now we create a write using the getWriter() method to write data in the stream. So the writer writes data in the chunks and after completing the write operation it closes the stream.

```
<!DOCTYPE html>
<html>
<body>
<script>
   // Creating a writable stream
   const writStream = new WritableStream({
      // processing the received chunks
      write(chunk) {
         console.log('Received chunk:', chunk);
      },
      // Closing the stream
      close(){
         console.log('Stream is closed');
      },
      // Handling the aborting stream
      abort(reason){
         console.log('Stream is aborted:', reason);
      }
   });
   // Create a writer to write in the stream
   const myWriter = writStream.getWriter();
```

```
    // Writing in the stream
    myWriter.write('Orange');
    myWriter.write('Blue');
    myWriter.write('Pink');
    myWriter.write('Red');

    // Close the stream
    myWriter.close();
</script>
</body>
</html>
```

Output



Conclusion

So this is a writable stream. With the help of writable, we can easily write data to the resources without loading the entire data in the memory. Now in the next article, we will discuss transform streams in Stream API.