Stream API - Transform Streams

In Stream API, transform streams are used to implement the pipe chain concept. The pipe chain is a process in which multiple streams are connected with each other. The original source is known as the starting point of the pipe chain whereas the ultimate sink is known as the ending point of the pipe chain.

TransformStream Interface

Stream API supports two types of transform stream interfaces –

- TransformStream Interface
- TransformStreamDefaultController

TransformStream Interface

The TransformStream Interface is used to implement the pipe chain transform stream method.

Constructor

To create a transform stream object, the TransformStream interface provides a TransformStream() constructor. This object represents the pair of streams that are WritableStream for the writable side and ReadableStream for the readable side.

Syntax

```
const newTrans = new TransformStream()
Or
const newTrans = new TransformStream(mtransform)
Or
const newTrans = new TransformStream(mtransform, writableStrategy)
```

```
Or
const newTrans = new TransformStream(mtransform, writableStrategy, readableStrate
```

Following are the optional parameters of the TransformStream() constructor –

- **mtransform** – This object represent the transformer. start(controller), transform(chunk, controller) and flush(controller) are the method contain by the transformer object. Where the controller is the instance of TransformStreamDefaultController.
- **writableStrategy** – This object is used to define the queuing strategy for the write streams. It takes two parameters: highWaterMark, and size(chunk).
- **readableStrategy** – This object is used to define the queuing strategy for the read streams. It takes two parameters: highWaterMark, and size(chunk).

Instance Properties

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

Sr.No.	Property & Description
1	**TransformStream.readable** This property returns a readable end of the TransformStream.
2	**TransformStream.writable** This property returns a writable end of the TransformStream.

TransformStreamDefaultController Interface

The TransformStreamDefaultController interface provides various methods to manipulate the ReadableStream and WritableStream. When we create a TransformStream then the TransformStreamDefaultController is automatically created. So it does not require any separate constructor.

Instance Properties

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

Sr.No.	Property & Description

1	**TransformStreamDefaultController.desiredSize** This property returns a size that will fill the readable side of the internal queue of a stream.

Methods

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

Sr.No.	Method & Description
1	**TransformStreamDefaultController.enqueue()** This method is used to enqueue a piece of data on the readable side of the given stream.
2	**TransformStreamDefaultController.error()** This method is used to find the error on both the readable and writable sides of the stream.
3	**TransformStreamDefaultController.terminate()** This method is used to close the readable side and errors of the writeable side of the transform stream.

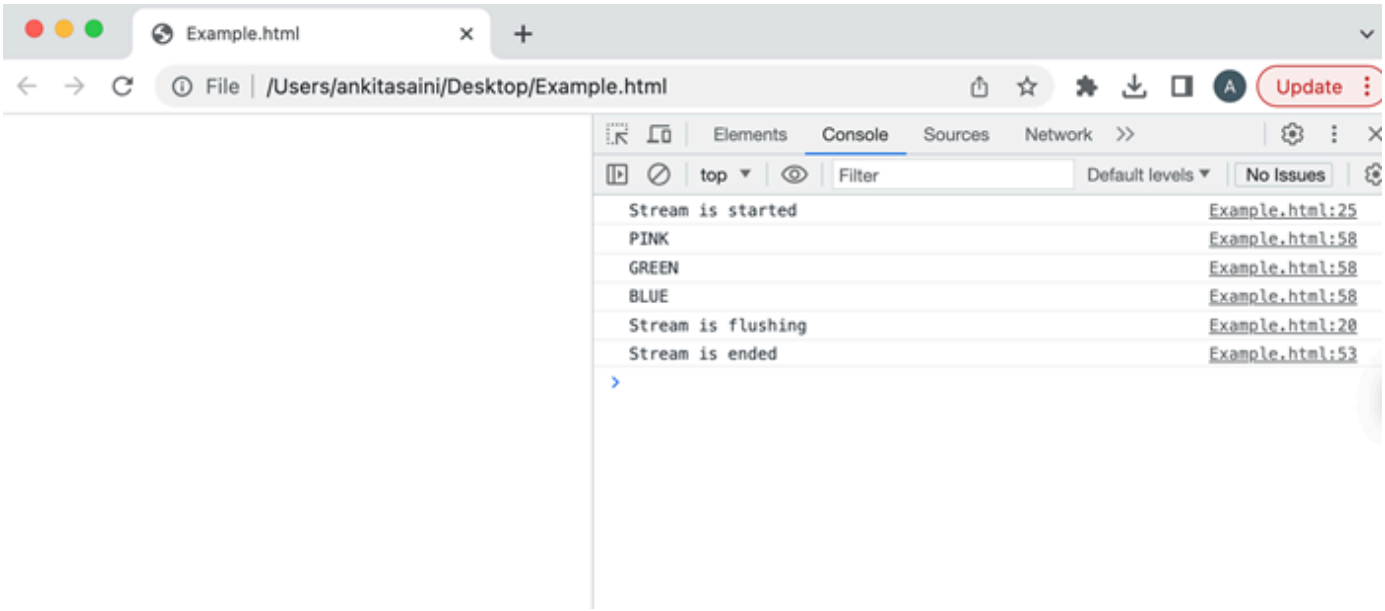
Example - Creating Transform Stream

In the following program, we create a custom transform stream. So to create a transform stream we use TransformStream() constructor with the transform(), flush(), start() and cancel() functions. The transform() function implement received chunks and then transform them in the uppercase and then enqueue the data using enqueue() method. The flush() method is to handle stream finalization, the start() method is used to handle initialization and the cancel() method is used to handle cancellation. Now we get the writer from the transform stream using the getWriter() method to read the data of the stream. Then we get the reader for the transform stream using the getReader() function. It reads and processes transformed data from the stream with the help of the myread() function.

```
<!DOCTYPE html>
<html>
<body>
<script>
   // Create a transform stream using TransformStream() constructor
   const newTransform = new TransformStream({
      transform(chunk, controller) {
         // Processing the received chunk in uppercase
```

```
         const tData = chunk.toString().toUpperCase();

         // Enqueue the transformed data and passed it to the downstream
         controller.enqueue(tData);
      },
      // Handling the finalized data, if required
      flush(controller) {
         console.log('Stream is flushing');
      },
      // Performing the initialization, if required
      start(controller) {
         console.log('Stream is started');
      },
      // Handling the stream if it is cancelled
      cancel(reason) {
         console.log('Stream got canceled:', reason);
      }
   });
   // Creating a writer for the transform stream
   const twriter = newTransform.writable.getWriter();

   // Writing the data into the transform stream
   twriter.write('pink');
   twriter.write('green');
   twriter.write('blue');

   // Closing the stream
   twriter.close();

   // Creating a reader for the transform stream
   const treader = newTransform.readable.getReader();

   // Read and process data from the transform stream
   function myread(){
      treader.read().then(({ done, value }) => {
         if (done) {
            console.log('Stream is ended');
            return;
         }
         // Processing the received transformed data
         console.log(value);
```

```
            // Continue reading data from the stream
            myread();
        });
    }
    // Calling the myread() to start reading from the transform stream
    myread();
</script>
</body>
</html>
```



Conclusion

So this is how the transform stream works. It is generally used when we connect multiple streams together. Now in the next article, we will learn about object mode in Stream API.