Uploading a Video

After generating the upload URL, you can now upload your video. This is done using the chunked upload method with retry handling.

> **Upload limits**
>
> The movingimage REST API has **a rate limit of 100 requests per minute**. Any requests that exceed this limit will be denied.

> The token in the upload URL is valid for four hours. This means that if the video upload takes longer than four hours to complete, an error will occur. If this happens, perform the "Get upload URL" request again to generate a new upload URL.

Chunked Upload

Sample

```
curl -X POST -H "Mi24-Upload-Total-Chunks: 10" -H "Mi24-Upload-Current-Chunk: 1" -H "Content-Type: application/octet-stream" --data-binary "@/<FILENAME>" "<UPLOAD_URL>"
```

URL	Description
UPLOAD_URL	The complete upload URL (see "Getting the Upload URL"). Note that the security token **must** be included as a parameter.
Headers	
Mi24-Upload-Total-Chunks	Total number of chunks. The example above uses 10 chunks.
Mi24-Upload-Current-Chunk	Current chunk number, starting with 1. The above example is the first chunk out of 10.
Content-Type	Specify "application/octet-stream" for the content type as in the example above.
Command-line Options	
--data-binary <@ /FILENAME>	Upload as bytes of binary data and be sure to prefix the filename with "@". Then indicate the filename (and path if necessary) of the local video file to upload.

> You would need to send the above example request 10 times, incrementing the current chunk each time. You can upload your file in as many or as few chunks as you wish, although typically the chunk size should be between 2 and 10MB. By using a larger number of smaller chunks, and implementing retry handling (see next section), you can expect a smoother and more reliable upload of your videos.

Response codes

`201 CREATED` Successful upload of an individual chunk. The entire upload is only complete when you receive this status for **all** chunks. See "Retry Handling" below if you do not receive this code for an uploaded chunk.

`200 OK` Video chunk has already been uploaded. This response code is returned when the chunk has already been successfully uploaded.

`4XX ERROR` Client-side error. Response code for when the server rejects a client request due to authorization errors, authentication errors or badly-formed requests.

`5XX ERROR` Server-side error. Response codes for when the server was unable to process a request.

`1101` If you receive this response, it means your upload was denied because you've reached the limits of your booked storage capacity

---

Retry Handling

If the response to a chunk upload request is not `200 OK` or `201 CREATED`, the chunk must be uploaded again.

This can happen for a few reasons, such as:

- Failure in the connection across the Internet
- Expired upload token

- Rate-limiting
- Timeout in receiving a response from the API

To avoid these situations interrupting your upload, we strongly suggest implementing retry handling in your code.

A good strategy for retry handling is to wait for an incrementally longer or random time between each failure. There are several options, for example:

- wait for 1 second, then 2, then 3 ... incrementing by one second each time
- wait for 1 second, then 2, then 3, 5, 8, 13 ... incrementing using a Fibonacci sequence
- wait for between 1 and 5 seconds, chosen randomly

Regardless of method, you should put a limit on the number of times you retry. Sometimes failure causes are more than transient and no amount of retries will succeed.

Java 11 Chunk Upload Example

```
package mi.uploader;

import java.io.*;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.time.Duration;

public class App {

    public static Integer CHUNK_SIZE = 2_097_152;
    public static Integer MAX_RETRIES = 10;

    public static HttpClient client = HttpClient.newBuilder()
            .version(HttpClient.Version.HTTP_1_1)
            .followRedirects(HttpClient.Redirect.NORMAL)
            .connectTimeout(Duration.ofSeconds(20))
            .build();

    public static void main(String[] args) throws IOException, InterruptedException {

        /*
         Step 1. https://doc.movingimage.com/display/LT/Creating+a+Video+Entity
         Step 2. https://doc.movingimage.com/display/LT/Getting+the+Upload+URL
         Step 3. Upload
         */

        URI url = URI.create("{URL obtained via API}");
        File f = new File("sample_600s_25fps_1080.mp4");
        InputStream inputStream = new FileInputStream(f);

        long totalChunks = calculateChunks(f.length());


        for (int i = 1; i <= totalChunks; i++) {

            byte[] data = new byte[CHUNK_SIZE];

            inputStream.read(data);

            int statusCode = 0;
            int retryAttempts = 0;
            do {
                if (statusCode >= 500) {
                    Thread.sleep(2000);
                }

                if (retryAttempts++ >= MAX_RETRIES) {
                    throw new RuntimeException("Upload failed. Please try again later.");
                }

                HttpRequest request = HttpRequest.newBuilder()
                        .POST(HttpRequest.BodyPublishers.ofByteArray(data))
                        .uri(url)
                        .setHeader("Mi24-Upload-Total-Chunks", String.valueOf(totalChunks)) // add
request header
                        .setHeader("Mi24-Upload-Current-Chunk", String.valueOf(i)) // add request header
```

```
                        .build();

                HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.
ofString());

                statusCode = response.statusCode();

                System.out.println("Status: "+statusCode);

            } while (statusCode >= 500);

        }

        System.out.println("Done");

    }

    public static long calculateChunks(long fileSize) {

        long remainder = fileSize % CHUNK_SIZE;
        int totalChunks = Math.toIntExact(fileSize / CHUNK_SIZE);

        if (remainder > 0) {
            totalChunks = totalChunks + 1;
        }

        return totalChunks;
    }
}
```