Media streaming basics

- Streaming use cases
- Streaming protocols
  - Using <video> to play a stream
  - Media Source Extensions
  - Basic example
- Media chunks format

In this article, you are going to learn about the more advanced concept of media streaming and by the end should have a good understanding of the various streaming use cases, protocols, and extensions. Let's start with an explanation of what streaming actually is.

Media streaming is a way of delivering and playing back media content piece by piece. Instead of loading a single file, which can be slow if not optimized for the network, the player reads a manifest file describing how the target media is split into individual chunks of data. Media chunks are later dynamically stitched back together at runtime—probably at different bitrates, which you'll learn about later.

Keep in mind that to provide streaming on your website the server must support the Range HTTP request header. Learn more about the `Accept-Ranges` header in The <video> and <source> tags article.

Streaming use cases

Producing media chunks and the necessary manifests describing the stream is not exactly straightforward, but streaming unlocks some interesting use cases that are not possible to achieve just by pointing a `<video>` element to a set of static source files. You'll learn more about how to add media to a web page in a later section. First, you should know about a few use cases for streaming multimedia if you want to go further than just loading multiple files into the `<video>` element.

- **Adaptive streaming** is where media chunks are encoded in several bitrates, and the highest quality media chunk that ***fits*** the client's currently available bandwidth is returned to the media player.
- **Live broadcast** is where media chunks are encoded and made available in real time.
- **Injecting media** is where other media like advertisements are injected into a stream without the player having to change the media source.

Streaming protocols

The two most commonly used streaming protocols on the web are **Dynamic Adaptive Streaming over HTTP** (DASH) and **HTTP Live Streaming** (HLS). Players that support these protocols will fetch the generated manifest file, figure out which media chunks to request, and then combine them into the final media experience.

Using `<video>` to play a stream

Many browsers are not going to play your stream natively. While there is some native support for HLS playback, browsers generally don't support native DASH stream playback. This means often it's not enough to simply point the `<source>` in the `<video>` element to a manifest file.

```
<video controls>
  <source src="manifest.mpd" type="application/dash+xml">
</video>
```

{% Aside 'caution' %} This is valid HTML, but doesn't actually work. Browsers don't natively support DASH manifest playback added to the `src` property. {% endAside %}

What may seem as a deficit is actually a strength in disguise. Streams are powerful and applications that consume streams have different needs.

Manifest files usually describe many variants of single media. Think different bitrates, several audio tracks, and even the same media encoded in different formats.

Some applications may want to keep a larger amount of video in the buffer, others may want to prefetch the first few seconds of video from an upcoming episode, and some want to implement their own logic for adaptive streaming. This is where you would want to have some sort of built-in browser feature to generate media streams for playback, and it just so happens there is one.

Media Source Extensions

Thankfully, the W3C defined something called Media Source Extensions (MSE) that will let JavaScript generate our media streams. In a nutshell, MSE allows developers to attach a `MediaSource` object to a `<video>` element and have it play back whatever media data is pumped into the buffers attached to the `MediaSource` instance.

Basic example

```
const videoEl = document.querySelector('video');
const mediaSource = new MediaSource();

video.src = URL.createObjectURL(mediaSource);
mediaSource.addEventListener(
  'sourceopen',
  () => {
    const mimeString = 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"';
    const buffer = mediaSource.addSourceBuffer(mimeString);

    buffer.appendBuffer( /* Video data as `ArrayBuffer` object. */ )
  }
);
```

The simplified example above illustrates a few things:

- As far as `<video>` is concerned, it is receiving media data from a URL.
- The generated URL is just a pointer to a `MediaSource` instance.
- The `MediaSource` instance creates one or more `SourceBuffer` instances.
- We then just append binary media data into the buffer, e.g. using `fetch`.

While these basic concepts are simple, and it is certainly possible to write a DASH and HLS compatible video player from scratch, most people usually pick one of the mature open source solutions that already exist, such as Shaka Player, JW Player, or Video.js to name a few.

However, we have created a demo Media PWA called Kino that demonstrates how you would go about developing your own basic streaming media website that provides offline media playback using just the simple `<video>` element. There are plans in our roadmap to support frameworks and digital rights management, among other features. So check back for updates from time to time, or request a feature. Read more about it in the PWA with offline streaming article.

Media chunks format

For a long time, DASH and HLS required media chunks to be encoded in different formats. In 2016, however, support for standard **fragmented MP4** (fMP4) files was added to HLS, a format that DASH also supports.

Video chunks using the `fMP4` container and the `H.264` codec are supported by both protocols and playable by a vast majority of players. This allows content producers to encode their videos just once, which in turn **saves time and disk space**.

To achieve better quality and lower files sizes, you may want to choose to encode several sets of media chunks using more efficient formats like `VP9`, though before we get to far ahead you will first need to learn how to Prepare media files for the web, and that's up next.

□□□ #2
□ □□□ □□□ 2026-10-31 22:56:55 UTC
□ □□□ □□□ 2026-10-31 22:56:56 UTC