

Base64

Base64 is a group of similar [binary-to-text encoding](#) schemes that represent binary data in an [ASCII](#) string format by translating it into a radix-64 representation. The term *Base64* originates from a specific [MIME content transfer encoding](#).

When the term "Base64" is used on its own to refer to a specific algorithm, it typically refers to the version of Base64 outlined in [RFC 4648](#), section 4, which uses the following alphabet to represent the radix-64 digits, alongside `=` as a padding character:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/_
```

A common variant is "Base64 URL safe", which omits the padding and replaces `+/_` with `-_` to avoid characters that might cause problems in URL path segments or query parameters.

Base64 encoding schemes are commonly used to encode binary data for storage or transfer over media that can only deal with ASCII text (or some superset of ASCII that still falls short of accepting arbitrary binary data). This ensures that the data remains intact without modification during transport. Common applications of Base64 include:

- Email via [MIME](#)
- Storing complex data in [XML](#)
- Encoding binary data so it can be included in a [data: URL](#)

Encoded size increase

Each Base64 digit represents 6 bits of data. So, three 8-bit bytes of the input string/binary file (3×8 bits = 24 bits) can be represented by four 6-bit Base64 digits (4×6 = 24 bits).

This means that the Base64 version of a string or file is typically roughly a third larger than its source (the exact size increase depends on various factors, such as the absolute length of the string, its length modulo 3, and whether padding characters are used).

JavaScript support

Browsers natively provide two JavaScript functions for decoding and encoding Base64 strings:

- [btoa](#) : creates a Base64-encoded ASCII string from a string of binary data ("btoa" should be read as "binary to ASCII").
- [atob](#) : decodes a Base64-encoded string ("atob" should be read as "ASCII to binary").

Note: Base64 is a binary encoding rather than a text encoding, but `btoa` and `atob` were added to the web platform before it supported binary data types. As a result, the two functions use strings to represent binary data, with the code point of each character representing the value of each byte. This has led to a common misconception that `btoa` can be used to encode arbitrary text data — for example, creating a Base64 `data:` URL of a text or HTML document.

However, the byte-to-code-point correspondence only reliably holds true for code points up to `0x7f`. Furthermore, code points over `0xff` will cause `btoa` to throw an error due to exceeding the maximum value for 1 byte. The next section details how to work around this limitation when encoding arbitrary Unicode text.

The "Unicode Problem"

Since `btoa` interprets the code points of its input string as byte values, calling `btoa` on a string will cause a "Character Out Of Range" exception if a character's code point exceeds `0xff`. For use cases where you need to encode arbitrary Unicode text, it is necessary to first convert the string to its constituent bytes in UTF-8, and then encode the bytes.

The simplest solution is to use `TextEncoder` and `TextDecoder` to convert between UTF-8 and single-byte representations of the string:

JS

```
function base64ToBytes(base64) {
  const binString = atob(base64);
  return Uint8Array.from(binString, (m) => m.codePointAt(0));
}

function bytesToBase64(bytes) {
  const binString = Array.from(bytes, (x) => String.fromCharCode(x)).join("");
  return btoa(binString);
}

// Usage
bytesToBase64(new TextEncoder().encode("a Ã ¤ 文 🐱")); // "YSDEgCDwkICAI0aWhyDwn6aE"
new TextDecoder().decode(base64ToBytes("YSDEgCDwkICAI0aWhyDwn6aE")); // "a Ã ¤ 文 🐱"
```

Converting arbitrary binary data

The `bytesToBase64` and `base64ToBytes` functions in the previous section can be used directly to convert between Base64 strings and [Uint8Array](#)s.

Alternatively, asynchronous conversion between base64 data URLs is possible natively within the web platform via the [FileReader](#) and [fetch](#) APIs:

JS

```
async function bytesToBase64DataURL(bytes, type = "application/octet-stream") {
  return await new Promise((resolve, reject) => {
    const reader = Object.assign(new FileReader(), {
      onload: () => resolve(reader.result),
      onerror: () => reject(reader.error),
    });
    reader.readAsDataURL(new File([bytes], "", { type }));
  });
}

async function dataUrlToBytes(dataUrl) {
  const res = await fetch(dataUrl);
  return new Uint8Array(await res.arrayBuffer());
}
```

// Usage

```
await bytesToBase64DataUrl(new Uint8Array([0, 1, 2])); // "data:application/octet-  
stream;base64,AAEC"  
await dataUrlToBytes("data:application/octet-stream;base64,AAEC"); // Uint8Array [0, 1, 2]
```

This page was last modified on Aug 30, 2023 by [MDN contributors](#).