

# URL

- [URL](#)
- [URL](#)
- [URL](#)
  - [URL.createObjectURL\(\)](#)
  - [URL.revokeObjectURL\(\)](#)
  - [URL.canParse\(\)](#)
  - [URL.parse\(\)](#)
- [URL](#)
  - [toString\(\)](#)

`URL` `URL`

`URL`

`URL` `URL`

`URL`

```
let url = new URL('https://example.com');
```

`URL` `try...catch` `URL`

`URL` `/foo/index.html` `URL`

```
const url1 = new URL('page2.html', 'http://example.com/page1.html');
url1.href // "http://example.com/page2.html"

const url2 = new URL('..', 'http://example.com/a/b.html')
url2.href // "http://example.com/"
```

`URL` `URL`

URL()

URL

URL()

href



URL

URL

- href
- protocol :
- search ?
- hash #
- username
- password
- host
- hostname
- port
- origin
- pathname /
- searchParams URLSearchParams

```
const url = new URL('http://user:pass@example.com:8080/resource/path?q=1#hash');
```

```
url.href // http://user:pass@example.com:8080/resource/path?q=1#hash
```

```
url.protocol // http:
```

```
url.username // user
```

```
url.password // pass
```

```
url.host // example.com:8080
```

```
url.hostname // example.com
```

```
url.port // 8080
```

```
url.pathname // /resource/path
```

```
url.search // ?q=1
```

```
url.hash // #hash
```

```
url.origin // http://example.com:8080
```

origin

```
const url = new URL('http://example.com/index.html#part1');
```

```
url.pathname = 'index2.html';
```

```
url.href // "http://example.com/index2.html#part1"
```

```
url.hash = '#part2';
url.href // "http://example.com/index2.html#part2"
```

URL URL pathname hash URL

searchParams URLSearchParams

```
const url = new URL('http://example.com/path?a=1&b=2');

url.searchParams.get('a') // 1
url.searchParams.get('b') // 2

for (const [k, v] of url.searchParams) {
  console.log(k, v);
}
// a 1
// b 2
```



## URL.createObjectURL()

URL.createObjectURL() URL Blob

```
// HTML
// <div id="display">
//   <input
//     type="file"
//     id="fileElem"
//     multiple
//     accept="image/*"
//     onchange="handleFiles(this.files)"
//   >
const div = document.getElementById('display');
```

```
function handleFiles(files) {
  for (let i = 0; i < files.length; i++) {
    let img = document.createElement('img');
    img.src = window.URL.createObjectURL(files[i]);
    div.appendChild(img);
  }
}
```

blob:http://localhost/c745ef73-ece9-46da-8f66-ebes574789b1

```
<script id='code' type='text/plain'>
  postMessage('foo');
</script>
<script>
  var code = document.getElementById('code').textContent;
  var blob = new Blob([code], { type: 'application/javascript' });
  var url = URL.createObjectURL(blob);
  var worker = new Worker(url);
  URL.revokeObjectURL(url);

  worker.onmessage = function(e) {
    console.log('worker returned: ', e.data);
  };
</script>
```

## URL.revokeObjectURL()

```
var div = document.getElementById('display');

function handleFiles(files) {
  for (var i = 0; i < files.length; i++) {
    var img = document.createElement('img');
    img.src = window.URL.createObjectURL(files[i]);
    div.appendChild(img);
    img.onload = function() {
      window.URL.revokeObjectURL(this.src);
    }
  }
}
```

The diagram illustrates the process of loading an image. It starts with a URL, represented by a box containing 'URL.revokeObjectURL()'. An arrow points from this box to another box containing 'img.onload'. This indicates that the image loading process is triggered by the URL and results in the 'img.onload' event being fired.

# URL.canParse()

```
URL() [ ] try...catch  
[ ] URL [ ] URL.canParse()  
[ ]
```

```
URL.canParse(url)
URL.canParse(url, base)
```

URL.canParse() [ ][ ][ ][ ][ ][ ][ ]

- [illegible]

```
URL.canParse("https://developer.mozilla.org/") // true
URL.canParse("/en-US/docs") // false
URL.canParse("/en-US/docs", "https://developer.mozilla.org/") // true
```

☐ URL ☐ false

\_\_\_\_\_ URL \_\_\_\_\_

```
let baseUrl = new URL("https://developer.mozilla.org/");  
let url = "/en-US/docs";
```

```
URL.canParse(url, baseUrl) // true
```

URL

URL() URL

URL() URL

```
function isValidUrl(string) {
  try {
    new URL(string);
    return true;
  } catch (err) {
    return false;
  }
}
```

URL

URL.canParse() URL

isValidUrl()

## URL.parse()

URL.parse() URL

Chromium 126 Firefox 126 URL

URL

URL() URL

URL

URL

URL

null

```
const urlString = "this is not a URL";

const not_a_url = URL.parse(urlstring); // null
```

URL

URL.parse() URL

null

URL

## toString()

URL URL

toString() URL

URL.href URL

URL #1

2023-09-11 22:15:28 UTC

2025-10-13 14:01:43 UTC