

Url

Url of Derafu Support, the canonical form of the paths of URLs

Anonymous · 08/10/2026

Url

`Url` works with the **paths** of URLs. It is made of pure functions that depend only on their arguments, and it works on a path, not on a whole URL: a `?` or a `#` in it is a character of the path, as any other.

Why a canonical form

A path can be written in many ways that mean the same thing for whoever reads it: `/api//index`, `/api/.index`, `/api/%69index` and `/api/index/` are all `/api/index`. When a component decides by the **text** of the path (a rule that protects `/api/index`) and another one reads its **meaning** (a router, a program that serves files), they must agree on which is the path. If they do not, what the first one does not recognize gets to the second one: `/api//index` is not under the rule `/api/index`, but the API serves it as if it was.

The functions of `Url` give one canonical form for each path, so every component can use the same one.

The canonical form

```
use Derafu\Support\Url;

Url::normalizePath('/api/index');           // "/api/index"
Url::normalizePath('//api//index/');        // "/api/index"
Url::normalizePath('/api/.index');          // "/api/index"
Url::normalizePath('/api/%69index');        // "/api/index"
Url::normalizePath('/ma%c3%blana');         // "/ma%C3%B1ana"
Url::normalizePath('');                     // "/"
```

The canonical form:

- starts with `/`, and has no empty segments (`//`), no `.` segments and no `/` at the end (except for the root, `/`);
- has the escapes of the characters that need no escape (letters, numbers and `-`, `.`, `_`, `~`) decoded (`%69` is `i`), and the hexadecimal digits of the other escapes in capitals (`%c3%b1` is `%C3%B1`), as [RFC 3986](#) says in section 6.2.2;

- keeps the case of the letters, and everything else as it is: `%20` stays encoded, and `%25` is not decoded again, so `%252F` is not a `/`.

What has no safe form

A path that can not be written in a canonical form without changing what it means gives `null`:

The path has	Example
A <code>..</code> segment, also escaped	<code>/a/./b</code> , <code>/a/%2e%2e/b</code>
A separator hidden in an escape, or a backslash	<code>/a%2Fb</code> , <code>/a%5Cb</code> , <code>/a\b</code>
A control character, also escaped, the null byte among them	<code>/a%00b</code> , <code>/a%1Fb</code>
An escape that is not valid	<code>/a%zz</code> , <code>/a%4</code> , <code>/a%</code>

A separator in an escape is not accepted because a component that decodes it would see more segments than the one that did not. As with [Ip](#), a path that has no safe form is a question that has an answer: the functions say `null` or `false`, they do not throw. Whoever receives requests decides what to do with it (refuse it, for example).

Functions

Method	What it does
<code>Url::normalizePath(\$path)</code>	The canonical form of the path, or <code>null</code> .
<code>Url::isNormalizedPath(\$path)</code>	Whether the path is its own canonical form (<code>false</code> if it has none).
<code>Url::pathSegments(\$path)</code>	The segments of the canonical form (none for the root), or <code>null</code> .
<code>Url::pathStartsWith(\$path, \$prefix, \$caseSensitive = true)</code>	Whether the path is the prefix or is below it, by segments.

```
Url::isNormalizedPath('/api/index');    // true
Url::isNormalizedPath('/api//index');  // false
Url::pathSegments('//api/./%69dex/');   // ['api', 'index']
```

Prefixes

`pathStartsWith()` compares by **segments**, and both the path and the prefix in their canonical form, so the way they are written does not matter. `/api` is the prefix of `/api`, of `/api/index` and of `/api/index/x`, and it is **not** the prefix of `/apiary`. The root, `/`, is the prefix of every path.

```
Url::pathStartsWith('/api/index', '/api');    // true
```

```
Url::pathStartsWith('/api', '/api');           // true
Url::pathStartsWith('/apiary', '/api');         // false: not a segment boundary.
Url::pathStartsWith('/api//index', '/api/index'); // true: the same path, written
another way.
Url::pathStartsWith('/api/%69index', '/api/index'); // true
Url::pathStartsWith('/api/index', '/');         // true
Url::pathStartsWith('/api/./x', '/api');        // false: it has no safe form.
```

It is `false` if the path or the prefix has no safe form: a rule that is not valid does not match anything, so whoever uses it as a rule has to check that the prefix is valid when it is configured.

By default the case matters (`/API` is not `/api`), as it does for most servers. A rule that **protects** a path must not depend on it: a server that reads files from a file system that does not tell them apart (the default one of macOS and Windows) serves `/Academy/x` with the file of `/academy/x`. For that, use `$caseSensitive = false`, which keeps the boundary of the segments:

```
Url::pathStartsWith('/API/Index', '/api', caseSensitive: false); // true
Url::pathStartsWith('/APIARY', '/api', caseSensitive: false);    // false
```

Last updated on 08/10/2026