

# **1 APR::URI - Perl API for URI manipulations**

## 1.1 Synopsis

```

use APR::URI ();

my $url = 'http://user:pass@example.com:80/foo?bar#item5';

# parse and break the url into components
my $parsed = APR::URI->parse($r->pool, $url);
print $parsed->scheme;
print $parsed->user;
print $parsed->password;
print $parsed->hostname;
print $parsed->port;
print $parsed->path;
print $parsed->rpath;
print $parsed->query;
print $parsed->fragment;

# reconstruct the url, after changing some components and completely
# removing other
$parsed->scheme($new_scheme);
$parsed->user(undef);
$parsed->password(undef);
$parsed->hostname($new_hostname);
$parsed->port($new_port);
$parsed->path($new_path);
$parsed->query(undef);
$parsed->fragment(undef);
print $parsed->unparse;

# get the password field too (by default it's not revealed)
use APR::Const -compile => qw(URI_UNP_REVEALPASSWORD);
print $parsed->unparse(APR::Const::URI_UNP_REVEALPASSWORD);

# what the default port for the ftp protocol?
my $ftp_port = APR::URI::port_of_scheme("ftp");

```

## 1.2 Description

APR::URI allows you to parse URI strings, manipulate each of the URI elements and deparse them back into URIs.

All APR::URI object accessors accept a string or an undef value as an argument. Same goes for return value. It's important to distinguish between an empty string and undef. For example let's say your code was:

```

my $uri = 'http://example.com/foo?bar#item5';
my $parsed = APR::URI->parse($r->pool, $uri);

```

Now you no longer want to the query and fragment components in the final url. If you do:

```
$parsed->fragment('');
$parsed->query('');
```

followed by:

```
my $new_uri = $parsed->unparse;
```

the resulting URI will be:

```
http://example.com/foo?#
```

which is probably not something that you've expected. In order to get rid of the separators, you must completely unset the fields you don't want to see. So, if you do:

```
$parsed->fragment(undef);
$parsed->query(undef);
```

followed by:

```
my $new_uri = $parsed->unparse;
```

the resulting URI will be:

```
http://example.com/foo
```

As mentioned earlier the same goes for return values, so continuing this example:

```
my $new_fragment = $parsed->fragment();
my $new_query    = $parsed->query();
```

Both values now contain `undef`, therefore you must be careful when using the return values, when you use them, as you may get warnings.

Also make sure you read through the `unparse()` section as various optional flags affect how the deparsed URI is rendered.

## 1.3 API

APR::URI provides the following functions and/or methods:

### 1.3.1 *fragment*

Get/set trailing "#fragment" string

```
$oldval = $parsed->fragment($newval);
```

- **obj:** `$parsed` (APR::URI object)
- **opt arg1:** `$newval` (string or undef)
- **ret:** `$oldval` (string or undef)
- **since:** 2.0.00

## 1.3.2 *hostinfo*

Get/set combined [user[:password]@]host[:port]

```
$oldval = $parsed->hostinfo($newval);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (string or undef)`
- **ret:** `$oldval (string or undef)`
- **since:** 2.0.00

The `hostinfo` value is set automatically when `parse()` is called.

It's not updated if any of the individual fields is modified.

It's not used when `unparse()` is called.

## 1.3.3 *hostname*

Get/set hostname

```
$oldval = $parsed->hostname($newval);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (string or undef)`
- **ret:** `$oldval (string or undef)`
- **since:** 2.0.00

## 1.3.4 *password*

Get/set password (as in `http://user:password@host:port/`)

```
$oldval = $parsed->password($newval);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (string or undef)`
- **ret:** `$oldval (string or undef)`
- **since:** 2.0.00

## 1.3.5 *parse*

Parse the URI string into URI components

```
$parsed = APR::URI->parse($pool, $uri);
```

- **obj:** `$parsed (APR::URI object or class)`
- **arg1:** `$pool (string) (APR::Pool object)`
- **arg2:** `$uri (string)`

The URI to parse

- **ret:** `$parsed (APR::URI object or class)`

The parsed URI object

- **since:** 2.0.00

After parsing, if a component existed but was an empty string (e.g. empty query `http://hostname/path?`) -- the corresponding accessor will return an empty string. If a component didn't exist (e.g. no query part `http://hostname/path`) -- the corresponding accessor will return undef.

## 1.3.6 path

Get/set the request path

```
$oldval = $parsed->path($newval);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (string or undef)`
- **ret:** `$oldval (string or undef)`

`"/` if only `scheme://host`

- **since:** 2.0.00

## 1.3.7 rpath

Gets the path minus the `path_info`

```
$rpath = $parsed->rpath();
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (string or undef)`
- **ret:** `$oldval (string or undef)`

The path minus the `path_info`

- **since:** 2.0.00

## 1.3.8 port

Get/set port number

```
$oldval = $parsed->port($newval);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (number or string or undef)`
- **ret:** `$oldval (string or undef)`

If the port component didn't appear in the parsed URI, APR internally calls `port_of_scheme()` to find out the port number for the given `scheme()`.

- **since:** 2.0.00

## 1.3.9 port\_of\_scheme

Return the default port for a given scheme. The recognized schemes are http, ftp, https, gopher, wais, nntp, snews and prospero.

```
$port = APR::URI::port_of_scheme($scheme);
```

- **obj:** `$scheme (string)`

The scheme string

- **ret:** `$port (integer)`

The default port for this scheme

- **since:** 2.0.00

## 1.3.10 query

Get/set the query string (the part starting after '?' and all the way till the end or the '#fragment' part if the latter exists).

```
$oldval = $parsed->query($newval);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (string or undef)`
- **ret:** `$oldval (string or undef)`
- **since:** 2.0.00

### 1.3.11 *scheme*

Get/set the protocol scheme ("http", "ftp", ...)

```
$oldval = $parsed->scheme($newval);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (string or undef)`
- **ret:** `$oldval (string or undef)`
- **since:** 2.0.00

### 1.3.12 *user*

Get/set user name (as in http://user:password@host:port/)

```
$oldval = $parsed->user($newval);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$newval (string or undef)`
- **ret:** `$oldval (string or undef)`
- **since:** 2.0.00

### 1.3.13 *unparse*

Unparse the URI components back into a URI string

```
$new_uri = $parsed->unparse();
$new_uri = $parsed->unparse($flags);
```

- **obj:** `$parsed (APR::URI object)`
- **opt arg1:** `$flags (the APR::Const::uri constants)`

By default the constant `APR::Const::URI_UNP_OMITPASSWORD` is passed.

If you need to pass more than one flag use unary `|`, e.g.:

```
$flags = APR::Const::URI_UNP_OMITUSER|APR::Const::URI_UNP_OMITPASSWORD;
```

The valid `flags` constants are listed next

- **ret:** `$new_uri (string)`
- **since:** 2.0.00

Valid `flags` constants:

To import all URI constants you could do:

```
use APR::Const -compile => qw(:uri);
```

but there is a significant amount of them, most irrelevant to this method. Therefore you probably don't want to do that. Instead specify explicitly the ones that you need. All the relevant to this method constants start with `APR::URI_UNP_`.

And the available constants are:

- **APR::Const::URI\_UNP\_OMITSITEPART**

Don't show `scheme`, `user`, `password`, `hostname` and `port` components (i.e. if you want only the relative URI)

- **APR::Const::URI\_UNP\_OMITUSER**

Hide the `user` component

- **APR::Const::URI\_UNP\_OMITPASSWORD**

Hide the `password` component (the default)

- **APR::Const::URI\_UNP\_REVEALPASSWORD**

Reveal the `password` component

- **APR::Const::URI\_UNP\_OMITPATHINFO**

Don't show `path`, `query` and `fragment` components

- **APR::Const::URI\_UNP\_OMITQUERY**

Don't show `query` and `fragment` components

Notice that some flags overlap.

If the optional `$flags` argument is passed and contains no `APR::Const::URI_UNP_OMITPASSWORD` and no `APR::Const::URI_UNP_REVEALPASSWORD` -- the `password` part will be rendered as a literal "XXXXXXXX" string.

If the `port` number matches the `port_of_scheme()`, the unparsed URI won't include it and there is no flag to force that `port` to appear. If the `port` number is non-standard it will show up in the unparsed string.

Examples:

Starting with the parsed URL:

```
use APR::URI ();
my $url = 'http://user:pass@example.com:80/foo?bar#item5';
my $parsed = APR::URI->parse($r->pool, $url);
```

deparse it back including and excluding parts, using different values for the optional `flags` argument:

- Show all but the password fields:

```
print $parsed->unparse;
```

Prints:

```
http://user@example.com/foo?bar#item5
```

Notice that the `port` field is gone too, since it was a default port for scheme `http://`.

- Include the password field (by default it's not revealed)

```
use APR::Const -compile => qw(URI_UNP_REVEALPASSWORD);
print $parsed->unparse(APR::Const::URI_UNP_REVEALPASSWORD);
```

Prints:

```
http://user:pass@example.com/foo?bar#item5
```

- Show all fields but the last three, path, query and fragment:

```
use APR::Const -compile => qw(URI_UNP_REVEALPASSWORD
                              APR::Const::URI_UNP_OMITPATHINFO);
print $parsed->unparse(
    APR::Const::URI_UNP_REVEALPASSWORD|URI_UNP_OMITPATHINFO);
```

Prints:

```
http://user:pass@example.com
```

## 1.4 See Also

Apache2::URI, `mod_perl 2.0` documentation.

## 1.5 Copyright

`mod_perl 2.0` and its core modules are copyrighted under The Apache Software License, Version 2.0.

## 1.6 Authors

The `mod_perl` development team and numerous contributors.



## Table of Contents:

1	APR::URI - Perl API for URI manipulations	1
1.1	Synopsis	2
1.2	Description	2
1.3	API	3
1.3.1	fragment	3
1.3.2	hostinfo	4
1.3.3	hostname	4
1.3.4	password	4
1.3.5	parse	4
1.3.6	path	5
1.3.7	rpath	5
1.3.8	port	6
1.3.9	port_of_scheme	6
1.3.10	query	6
1.3.11	scheme	7
1.3.12	user	7
1.3.13	unparse	7
1.4	See Also	9
1.5	Copyright	9
1.6	Authors	9