NAME

URI::file – URI that maps to local file names

SYNOPSIS

```
use URI::file;

$u1 = URI->new("file:/foo/bar");
$u2 = URI->new("foo/bar", "file");

$u3 = URI::file->new($path);
$u4 = URI::file->new("c:\\windows\\", "win32");

$u1->file;
$u1->file("mac");
```

DESCRIPTION

The `URI::file` class supports `URI` objects belonging to the *file* URI scheme. This scheme allows us to map the conventional file names found on various computer systems to the URI name space. An old specification of the *file* URI scheme is found in RFC 1738. Some older background information is also in RFC 1630. There are no newer specifications as far as I know.

If you simply want to construct *file* URI objects from URI strings, use the normal `URI` constructor. If you want to construct *file* URI objects from the actual file names used by various systems, then use one of the following `URI::file` constructors:

`$u` = URI::file->new(`$filename`, [$os])
: Maps a file name to the *file:* URI name space, creates a URI object and returns it. The `$filename` is interpreted as belonging to the indicated operating system ($os), which defaults to the value of the $^O variable. The `$filename` can be either absolute or relative, and the corresponding type of URI object for `$os` is returned.

`$u` = URI::file->new_abs(`$filename`, [$os])
: Same as URI::file->new, but makes sure that the URI returned represents an absolute file name. If the `$filename` argument is relative, then the name is resolved relative to the current directory, i.e. this constructor is really the same as:

```
URI::file->new($filename)->abs(URI::file->cwd);
```

`$u` = URI::file->cwd
: Returns a *file* URI that represents the current working directory. See Cwd.

The following methods are supported for *file* URI (in addition to the common and generic methods described in URI):

`$u`->file([$os])
: Returns a file name. It maps from the URI name space to the file name space of the indicated operating system.

  It might return `undef` if the name can not be represented in the indicated file system.

`$u`->dir([$os])
: Some systems use a different form for names of directories than for plain files. Use this method if you know you want to use the name for a directory.

The `URI::file` module can be used to map generic file names to names suitable for the current system. As such, it can work as a nice replacement for the `File::Spec` module. For instance, the following code translates the UNIX-style file name *Foo/Bar.pm* to a name suitable for the local system:

```
$file = URI::file->new("Foo/Bar.pm", "unix")->file;
die "Can't map filename Foo/Bar.pm for $^O" unless defined $file;
open(FILE, $file) || die "Can't open '$file': $!";
# do something with FILE
```

MAPPING NOTES

Most computer systems today have hierarchically organized file systems. Mapping the names used in these systems to the generic URI syntax allows us to work with relative file URIs that behave as they should when resolved using the generic algorithm for URIs (specified in RFC 2396). Mapping a file name to the generic URI syntax involves mapping the path separator character to "/" and encoding any reserved characters that appear in the path segments of the file name. If path segments consisting of the strings "." or ".." have a different meaning than what is specified for generic URIs, then these must be encoded as well.

If the file system has device, volume or drive specifications as the root of the name space, then it makes sense to map them to the authority field of the generic URI syntax. This makes sure that relative URIs can not be resolved "above" them, i.e. generally how relative file names work in those systems.

Another common use of the authority field is to encode the host on which this file name is valid. The host name "localhost" is special and generally has the same meaning as a missing or empty authority field. This use is in conflict with using it as a device specification, but can often be resolved for device specifications having characters not legal in plain host names.

File name to URI mapping in normally not one-to-one. There are usually many URIs that map to any given file name. For instance, an authority of "localhost" maps the same as a URI with a missing or empty authority.

Example 1: The Mac classic (Mac OS 9 and earlier) used ":" as path separator, but not in the same way as a generic URI. ":foo" was a relative name. "foo:bar" was an absolute name. Also, path segments could contain the "/" character as well as the literal "." or "..". So the mapping looks like this:

```
Mac classic           URI
----------            -------------------
:foo:bar     <==>     foo/bar
:            <==>     ./
::foo:bar    <==>     ../foo/bar
:::          <==>     ../../
foo:bar      <==>     file:/foo/bar
foo:bar:     <==>     file:/foo/bar/
..           <==>     %2E%2E
<undef>      <==      /
foo/         <==      file:/foo%2F
./foo.txt    <==      file:/.%2Ffoo.txt
```

Note that if you want a relative URL, you *must* begin the path with a :. Any path that begins with [^:] is treated as absolute.

Example 2: The UNIX file system is easy to map, as it uses the same path separator as URIs, has a single root, and segments of "." and ".." have the same meaning. URIs that have the character "\0" or "/" as part of any path segment can not be turned into valid UNIX file names.

```
UNIX                  URI
----------            ------------------
foo/bar      <==>     foo/bar
/foo/bar     <==>     file:/foo/bar
/foo/bar     <==      file://localhost/foo/bar
file:         ==>     ./file:
<undef>      <==      file:/fo%00/bar
/            <==>     file:/
```

CONFIGURATION VARIABLES

The following configuration variables influence how the class and its methods behave:

`%URI::file::OS_CLASS`

This hash maps OS identifiers to implementation classes. You might want to add or modify this if you want to plug in your own file handler class. Normally the keys should match the $^O values in use.

If there is no mapping then the ‘‘Unix’’ implementation is used.

`$URI::file::DEFAULT_AUTHORITY`

This determine what ‘‘authority’’ string to include in absolute file URIs. It defaults to "‘‘. If you prefer verbose URIs you might set it to be ’’localhost".

Setting this value to `undef` force behaviour compatible to URI v1.31 and earlier. In this mode host names in UNC paths and drive letters are mapped to the authority component on Windows, while we produce authority-less URIs on Unix.

SEE ALSO

URI, File::Spec, perlport

COPYRIGHT