import {Syntax} from './options';

import {PromiseOr} from './util/promise\_or';

/\*\*

\* A special type of importer that redirects all loads to existing files on

\* disk. Although this is less powerful than a full [[Importer]], it

\* automatically takes care of Sass features like resolving partials and file

\* extensions and of loading the file from disk.

\*

\* Like all importers, this implements custom Sass loading logic for [`@use`

\* rules](https://sass-lang.com/documentation/at-rules/use) and [`@import`

\* rules](https://sass-lang.com/documentation/at-rules/import). It can be passed

\* to [[Options.importers]] or [[StringOptionsWithImporter.importer]].

\*

\* @typeParam sync - A `FileImporter<'sync'>`'s [[findFileUrl]] must return

\* synchronously, but in return it can be passed to [[compile]] and

\* [[compileString]] in addition to [[compileAsync]] and [[compileStringAsync]].

\*

\* A `FileImporter<'async'>`'s [[findFileUrl]] may either return synchronously

\* or asynchronously, but it can only be used with [[compileAsync]] and

\* [[compileStringAsync]].

\*

\* @example

\*

\* ```js

\* const {pathToFileURL} = require('url');

\*

\* sass.compile('style.scss', {

\* importers: [{

\* // An importer that redirects relative URLs starting with "~" to

\* // `node\_modules`.

\* findFileUrl(url) {

\* if (!url.startsWith('~')) return null;

\* return new URL(url.substring(1), pathToFileURL('node\_modules'));

\* }

\* }]

\* });

\* ```

\*

\* @category Importer

\*/

export interface FileImporter<

sync extends 'sync' | 'async' = 'sync' | 'async'

> {

/\*\*

\* A callback that's called to partially resolve a load (such as

\* [`@use`](https://sass-lang.com/documentation/at-rules/use) or

\* [`@import`](https://sass-lang.com/documentation/at-rules/import)) to a file

\* on disk.

\*

\* Unlike an [[Importer]], the compiler will automatically handle relative

\* loads for a [[FileImporter]]. See [[Options.importers]] for more details on

\* the way loads are resolved.

\*

\* @param url - The loaded URL. Since this might be relative, it's represented

\* as a string rather than a [[URL]] object.

\*

\* @param options.fromImport - Whether this is being invoked because of a Sass

\* `@import` rule, as opposed to a `@use` or `@forward` rule.

\*

\* This should \*only\* be used for determining whether or not to load

\* [import-only files](https://sass-lang.com/documentation/at-rules/import#import-only-files).

\*

\* @returns An absolute `file:` URL if this importer recognizes the `url`.

\* This may be only partially resolved: the compiler will automatically look

\* for [partials](https://sass-lang.com/documentation/at-rules/use#partials),

\* [index files](https://sass-lang.com/documentation/at-rules/use#index-files),

\* and file extensions based on the returned URL. An importer may also return

\* a fully resolved URL if it so chooses.

\*

\* If this importer doesn't recognize the URL, it should return `null` instead

\* to allow other importers or {@link Options.loadPaths | load paths} to

\* handle it.

\*

\* This may also return a `Promise`, but if it does the importer may only be

\* passed to [[compileAsync]] and [[compileStringAsync]], not [[compile]] or

\* [[compileString]].

\*

\* @throws any - If this importer recognizes `url` but determines that it's

\* invalid, it may throw an exception that will be wrapped by Sass. If the

\* exception object has a `message` property, it will be used as the wrapped

\* exception's message; otherwise, the exception object's `toString()` will be

\* used. This means it's safe for importers to throw plain strings.

\*/

findFileUrl(

url: string,

options: {fromImport: boolean}

): PromiseOr<URL | null, sync>;

/\*\* @hidden \*/

canonicalize?: never;

}

/\*\*

\* An object that implements custom Sass loading logic for [`@use`

\* rules](https://sass-lang.com/documentation/at-rules/use) and [`@import`

\* rules](https://sass-lang.com/documentation/at-rules/import). It can be passed

\* to [[Options.importers]] or [[StringOptionsWithImporter.importer]].

\*

\* Importers that simply redirect to files on disk are encouraged to use the

\* [[FileImporter]] interface instead.

\*

\* See [[Options.importers]] for more details on the way loads are resolved.

\*

\* @typeParam sync - An `Importer<'sync'>`'s [[canonicalize]] and [[load]] must

\* return synchronously, but in return it can be passed to [[compile]] and

\* [[compileString]] in addition to [[compileAsync]] and [[compileStringAsync]].

\*

\* An `Importer<'async'>`'s [[canonicalize]] and [[load]] may either return

\* synchronously or asynchronously, but it can only be used with

\* [[compileAsync]] and [[compileStringAsync]].

\*

\* @example Resolving a Load

\*

\* This is the process of resolving a load using a custom importer:

\*

\* - The compiler encounters `@use "db:foo/bar/baz"`.

\* - It calls [[canonicalize]] with `"db:foo/bar/baz"`.

\* - [[canonicalize]] returns `new URL("db:foo/bar/baz/\_index.scss")`.

\* - If the compiler has already loaded a stylesheet with this canonical URL, it

\* re-uses the existing module.

\* - Otherwise, it calls [[load]] with `new URL("db:foo/bar/baz/\_index.scss")`.

\* - [[load]] returns an [[ImporterResult]] that the compiler uses as the

\* contents of the module.

\*

\* @example Code Sample

\*

\* ```js

\* sass.compile('style.scss', {

\* // An importer for URLs like `bgcolor:orange` that generates a

\* // stylesheet with the given background color.

\* importers: [{

\* canonicalize(url) {

\* if (!url.startsWith('bgcolor:')) return null;

\* return new URL(url);

\* },

\* load(canonicalUrl) {

\* return {

\* contents: `body {background-color: ${canonicalUrl.pathname}}`,

\* syntax: 'scss'

\* };

\* }

\* }]

\* });

\* ```

\*

\* @category Importer

\*/

export interface Importer<sync extends 'sync' | 'async' = 'sync' | 'async'> {

/\*\*

\* If `url` is recognized by this importer, returns its canonical format.

\*

\* If Sass has already loaded a stylesheet with the returned canonical URL, it

\* re-uses the existing parse tree (and the loaded module for `@use`). This

\* means that importers \*\*must ensure\*\* that the same canonical URL always

\* refers to the same stylesheet, \*even across different importers\*. As such,

\* importers are encouraged to use unique URL schemes to disambiguate between

\* one another.

\*

\* As much as possible, custom importers should canonicalize URLs the same way

\* as the built-in filesystem importer:

\*

\* - The importer should look for stylesheets by adding the prefix `\_` to the

\* URL's basename, and by adding the extensions `.sass` and `.scss` if the

\* URL doesn't already have one of those extensions. For example, if the

\* URL was `foo/bar/baz`, the importer would look for:

\* - `foo/bar/baz.sass`

\* - `foo/bar/baz.scss`

\* - `foo/bar/\_baz.sass`

\* - `foo/bar/\_baz.scss`

\*

\* If the URL was `foo/bar/baz.scss`, the importer would just look for:

\* - `foo/bar/baz.scss`

\* - `foo/bar/\_baz.scss`

\*

\* If the importer finds a stylesheet at more than one of these URLs, it

\* should throw an exception indicating that the URL is ambiguous. Note that

\* if the extension is explicitly specified, a stylesheet with the opposite

\* extension is allowed to exist.

\*

\* - If none of the possible paths is valid, the importer should perform the

\* same resolution on the URL followed by `/index`. In the example above,

\* it would look for:

\* - `foo/bar/baz/index.sass`

\* - `foo/bar/baz/index.scss`

\* - `foo/bar/baz/\_index.sass`

\* - `foo/bar/baz/\_index.scss`

\*

\* As above, if the importer finds a stylesheet at more than one of these

\* URLs, it should throw an exception indicating that the import is

\* ambiguous.

\*

\* If no stylesheets are found, the importer should return `null`.

\*

\* Calling [[canonicalize]] multiple times with the same URL must return the

\* same result. Calling [[canonicalize]] with a URL returned by a previous

\* call to [[canonicalize]] must return that URL.

\*

\* Relative loads in stylesheets loaded from an importer are handled by

\* resolving the loaded URL relative to the canonical URL of the stylesheet

\* that contains it, and passing that URL back to the importer's

\* [[canonicalize]] method. For example, suppose the "Resolving a Load"

\* example {@link Importer | above} returned a stylesheet that contained

\* `@use "mixins"`:

\*

\* - The compiler resolves the URL `mixins` relative to the current

\* stylesheet's canonical URL `db:foo/bar/baz/\_index.scss` to get

\* `db:foo/bar/baz/mixins`.

\* - It calls [[canonicalize]] with `"db:foo/bar/baz/mixins"`.

\* - [[canonicalize]] returns `new URL("db:foo/bar/baz/\_mixins.scss")`.

\*

\* Because of this, [[canonicalize]] must return a meaningful result when

\* called with a URL relative to one returned by an earlier call to

\* [[canonicalize]].

\*

\* @param url - The loaded URL. Since this might be relative, it's represented

\* as a string rather than a [[URL]] object.

\*

\* @param options.fromImport - Whether this is being invoked because of a Sass

\* `@import` rule, as opposed to a `@use` or `@forward` rule.

\*

\* This should \*only\* be used for determining whether or not to load

\* [import-only files](https://sass-lang.com/documentation/at-rules/import#import-only-files).

\*

\* @returns An absolute URL if this importer recognizes the `url`, or `null`

\* if it doesn't. If this returns `null`, other importers or {@link

\* Options.loadPaths | load paths} may handle the load.

\*

\* This may also return a `Promise`, but if it does the importer may only be

\* passed to [[compileAsync]] and [[compileStringAsync]], not [[compile]] or

\* [[compileString]].

\*

\* @throws any - If this importer recognizes `url` but determines that it's

\* invalid, it may throw an exception that will be wrapped by Sass. If the

\* exception object has a `message` property, it will be used as the wrapped

\* exception's message; otherwise, the exception object's `toString()` will be

\* used. This means it's safe for importers to throw plain strings.

\*/

canonicalize(

url: string,

options: {fromImport: boolean}

): PromiseOr<URL | null, sync>;

/\*\*

\* Loads the Sass text for the given `canonicalUrl`, or returns `null` if this

\* importer can't find the stylesheet it refers to.

\*

\* @param canonicalUrl - The canonical URL of the stylesheet to load. This is

\* guaranteed to come from a call to [[canonicalize]], although not every call

\* to [[canonicalize]] will result in a call to [[load]].

\*

\* @returns The contents of the stylesheet at `canonicalUrl` if it can be

\* loaded, or `null` if it can't.

\*

\* This may also return a `Promise`, but if it does the importer may only be

\* passed to [[compileAsync]] and [[compileStringAsync]], not [[compile]] or

\* [[compileString]].

\*

\* @throws any - If this importer finds a stylesheet at `url` but it fails to

\* load for some reason, or if `url` is uniquely associated with this importer

\* but doesn't refer to a real stylesheet, the importer may throw an exception

\* that will be wrapped by Sass. If the exception object has a `message`

\* property, it will be used as the wrapped exception's message; otherwise,

\* the exception object's `toString()` will be used. This means it's safe for

\* importers to throw plain strings.

\*/

load(canonicalUrl: URL): PromiseOr<ImporterResult | null, sync>;

/\*\* @hidden \*/

findFileUrl?: never;

}

/\*\*

\* The result of successfully loading a stylesheet with an [[Importer]].

\*

\* @category Importer

\*/

export interface ImporterResult {

/\*\* The contents of the stylesheet. \*/

contents: string;

/\*\* The syntax with which to parse [[contents]]. \*/

syntax: Syntax;

/\*\*

\* The URL to use to link to the loaded stylesheet's source code in source

\* maps. A `file:` URL is ideal because it's accessible to both browsers and

\* other build tools, but an `http:` URL is also acceptable.

\*

\* If this isn't set, it defaults to a `data:` URL that contains the contents

\* of the loaded stylesheet.

\*/

sourceMapUrl?: URL;

}