import {Logger} from '../logger';

import {LegacyImporter} from './importer';

import {LegacyFunction} from './function';

/\*\*

\* Options for [[render]] and [[renderSync]] that are shared between

\* [[LegacyFileOptions]] and [[LegacyStringOptions]].

\*

\* @typeParam sync - This lets the TypeScript checker verify that

\* [[LegacyAsyncImporter]]s and [[LegacyAsyncFunction]]s aren't passed to

\* [[renderSync]].

\*

\* @category Legacy

\* @deprecated This only works with the legacy [[render]] and [[renderSync]]

\* APIs. Use [[Options]] with [[compile]], [[compileString]], [[compileAsync]],

\* and [[compileStringAsync]] instead.

\*/

export interface LegacySharedOptions<sync extends 'sync' | 'async'> {

/\*\*

\* This array of strings option provides [load

\* paths](https://sass-lang.com/documentation/at-rules/import#load-paths) for

\* Sass to look for stylesheets. Earlier load paths will take precedence over

\* later ones.

\*

\* ```js

\* sass.renderSync({

\* file: "style.scss",

\* includePaths: ["node\_modules/bootstrap/dist/css"]

\* });

\* ```

\*

\* Load paths are also loaded from the `SASS\_PATH` environment variable, if

\* it’s set. This variable should be a list of paths separated by `;` (on

\* Windows) or `:` (on other operating systems). Load paths from the

\* `includePaths` option take precedence over load paths from `SASS\_PATH`.

\*

\* ```sh

\* $ SASS\_PATH=node\_modules/bootstrap/dist/css sass style.scss style.css

\* ```

\*

\* @category Input

\* @compatibility feature: "SASS\_PATH", dart: "1.15.0", node: "3.9.0"

\*

\* Earlier versions of Dart Sass and Node Sass didn’t support the `SASS\_PATH`

\* environment variable.

\*/

includePaths?: string[];

/\*\*

\* Whether the generated CSS should use spaces or tabs for indentation.

\*

\* ```js

\* const result = sass.renderSync({

\* file: "style.scss",

\* indentType: "tab",

\* indentWidth: 1

\* });

\*

\* result.css.toString();

\* // "h1 {\n\tfont-size: 40px;\n}\n"

\* ```

\*

\* @defaultValue `'space'`

\* @category Output

\* @compatibility dart: true, node: "3.0.0"

\*/

indentType?: 'space' | 'tab';

/\*\*

\* How many spaces or tabs (depending on [[indentType]]) should be used per

\* indentation level in the generated CSS. It must be between 0 and 10

\* (inclusive).

\*

\* @defaultValue `2`

\* @category Output

\* @compatibility dart: true, node: "3.0.0"

\*/

indentWidth?: number;

/\*\*

\* Which character sequence to use at the end of each line in the generated

\* CSS. It can have the following values:

\*

\* \* `'lf'` uses U+000A LINE FEED.

\* \* `'lfcr'` uses U+000A LINE FEED followed by U+000D CARRIAGE RETURN.

\* \* `'cr'` uses U+000D CARRIAGE RETURN.

\* \* `'crlf'` uses U+000D CARRIAGE RETURN followed by U+000A LINE FEED.

\*

\* @defaultValue `'lf'`

\* @category Output

\* @compatibility dart: true, node: "3.0.0"

\*/

linefeed?: 'cr' | 'crlf' | 'lf' | 'lfcr';

/\*\*

\* If `true`, Sass won't add a link from the generated CSS to the source map.

\*

\* ```js

\* const result = sass.renderSync({

\* file: "style.scss",

\* sourceMap: "out.map",

\* omitSourceMapUrl: true

\* })

\* console.log(result.css.toString());

\* // h1 {

\* // font-size: 40px;

\* // }

\* ```

\*

\* @defaultValue `false`

\* @category Source Maps

\*/

omitSourceMapUrl?: boolean;

/\*\*

\* The location that Sass expects the generated CSS to be saved to. It’s used

\* to determine the URL used to link from the generated CSS to the source map,

\* and from the source map to the Sass source files.

\*

\* \*\*Heads up!\*\* Despite the name, Sass does \*not\* write the CSS output to

\* this file. The caller must do that themselves.

\*

\* ```js

\* result = sass.renderSync({

\* file: "style.scss",

\* sourceMap: true,

\* outFile: "out.css"

\* })

\* console.log(result.css.toString());

\* // h1 {

\* // font-size: 40px;

\* // }

\* // /\*# sourceMappingURL=out.css.map \* /

\* ```

\*

\* @category Source Maps

\*/

outFile?: string;

/\*\*

\* The output style of the compiled CSS. There are four possible output styles:

\*

\* \* `"expanded"` (the default for Dart Sass) writes each selector and

\* declaration on its own line.

\*

\* \* `"compressed"` removes as many extra characters as possible, and writes

\* the entire stylesheet on a single line.

\*

\* \* `"nested"` (the default for Node Sass, not supported by Dart Sass)

\* indents CSS rules to match the nesting of the Sass source.

\*

\* \* `"compact"` (not supported by Dart Sass) puts each CSS rule on its own single line.

\*

\* @example

\*

\* ```js

\* const source = `

\* h1 {

\* font-size: 40px;

\* code {

\* font-face: Roboto Mono;

\* }

\* }`;

\*

\* let result = sass.renderSync({

\* data: source,

\* outputStyle: "expanded"

\* });

\* console.log(result.css.toString());

\* // h1 {

\* // font-size: 40px;

\* // }

\* // h1 code {

\* // font-face: Roboto Mono;

\* // }

\*

\* result = sass.renderSync({

\* data: source,

\* outputStyle: "compressed"

\* });

\* console.log(result.css.toString());

\* // h1{font-size:40px}h1 code{font-face:Roboto Mono}

\*

\* result = sass.renderSync({

\* data: source,

\* outputStyle: "nested"

\* });

\* console.log(result.css.toString());

\* // h1 {

\* // font-size: 40px; }

\* // h1 code {

\* // font-face: Roboto Mono; }

\*

\* result = sass.renderSync({

\* data: source,

\* outputStyle: "compact"

\* });

\* console.log(result.css.toString());

\* // h1 { font-size: 40px; }

\* // h1 code { font-face: Roboto Mono; }

\* ```

\*

\* @category Output

\*/

outputStyle?: 'compressed' | 'expanded' | 'nested' | 'compact';

/\*\*

\* Whether or not Sass should generate a source map. If it does, the source

\* map will be available as [[LegacyResult.map]] (unless [[sourceMapEmbed]] is

\* `true`).

\*

\* If this option is a string, it’s the path that the source map is expected

\* to be written to, which is used to link to the source map from the

\* generated CSS and to link \*from\* the source map to the Sass source files.

\* Note that if `sourceMap` is a string and [[outFile]] isn’t passed, Sass

\* assumes that the CSS will be written to the same directory as the file

\* option if it’s passed.

\*

\* If this option is `true`, the path is assumed to be [[outFile]] with `.map`

\* added to the end. If it’s `true` and [[outFile]] isn’t passed, it has no

\* effect.

\*

\* @example

\*

\* ```js

\* let result = sass.renderSync({

\* file: "style.scss",

\* sourceMap: "out.map"

\* })

\* console.log(result.css.toString());

\* // h1 {

\* // font-size: 40px;

\* // }

\* // /\*# sourceMappingURL=out.map \* /

\*

\* result = sass.renderSync({

\* file: "style.scss",

\* sourceMap: true,

\* outFile: "out.css"

\* })

\* console.log(result.css.toString());

\* // h1 {

\* // font-size: 40px;

\* // }

\* // /\*# sourceMappingURL=out.css.map \* /

\* ```

\*

\* @defaultValue `false`

\* @category Source Maps

\*/

sourceMap?: boolean | string;

/\*\*

\* Whether to embed the entire contents of the Sass files that contributed to

\* the generated CSS in the source map. This may produce very large source

\* maps, but it guarantees that the source will be available on any computer

\* no matter how the CSS is served.

\*

\* @example

\*

\* ```js

\* sass.renderSync({

\* file: "style.scss",

\* sourceMap: "out.map",

\* sourceMapContents: true

\* })

\* ```

\*

\* @defaultValue `false`

\* @category Source Maps

\*/

sourceMapContents?: boolean;

/\*\*

\* Whether to embed the contents of the source map file in the generated CSS,

\* rather than creating a separate file and linking to it from the CSS.

\*

\* @example

\*

\* ```js

\* sass.renderSync({

\* file: "style.scss",

\* sourceMap: "out.map",

\* sourceMapEmbed: true

\* });

\* ```

\*

\* @defaultValue `false`

\* @category Source Maps

\*/

sourceMapEmbed?: boolean;

/\*\*

\* If this is passed, it's prepended to all the links from the source map to

\* the Sass source files.

\*

\* @category Source Maps

\*/

sourceMapRoot?: string;

/\*\*

\* Additional handler(s) for loading files when a [`@use`

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

\* rule](https://sass-lang.com/documentation/at-rules/import) is encountered.

\* It can either be a single [[LegacyImporter]] function, or an array of

\* [[LegacyImporter]]s.

\*

\* Importers take the URL of the `@import` or `@use` rule and return a

\* [[LegacyImporterResult]] indicating how to handle that rule. For more

\* details, see [[LegacySyncImporter]] and [[LegacyAsyncImporter]].

\*

\* Loads are resolved by trying, in order:

\*

\* \* Loading a file from disk relative to the file in which the `@use` or

\* `@import` appeared.

\*

\* \* Each custom importer.

\*

\* \* Loading a file relative to the current working directory.

\*

\* \* Each load path in [[includePaths]].

\*

\* \* Each load path specified in the `SASS\_PATH` environment variable, which

\* should be semicolon-separated on Windows and colon-separated elsewhere.

\*

\* @example

\*

\* ```js

\* sass.render({

\* file: "style.scss",

\* importer: [

\* // This importer uses the synchronous API, and can be passed to either

\* // renderSync() or render().

\* function(url, prev) {

\* // This generates a stylesheet from scratch for `@use "big-headers"`.

\* if (url != "big-headers") return null;

\*

\* return {

\* contents: `

\* h1 {

\* font-size: 40px;

\* }`

\* };

\* },

\*

\* // This importer uses the asynchronous API, and can only be passed to

\* // render().

\* function(url, prev, done) {

\* // Convert `@use "foo/bar"` to "node\_modules/foo/sass/bar".

\* const components = url.split('/');

\* const innerPath = components.slice(1).join('/');

\* done({

\* file: `node\_modules/${components.first}/sass/${innerPath}`

\* });

\* }

\* ]

\* }, function(err, result) {

\* // ...

\* });

\* ```

\*

\* @category Plugins

\* @compatibility dart: true, node: "3.0.0"

\*

\* Versions of Node Sass before 3.0.0 don’t support arrays of importers, nor

\* do they support importers that return `Error` objects.

\*

\* Versions of Node Sass before 2.0.0 don’t support the `importer` option at

\* all.

\*

\* @compatibility feature: "Import order", dart: "1.20.2", node: false

\*

\* Versions of Dart Sass before 1.20.2 preferred resolving imports using

\* [[includePaths]] before resolving them using custom importers.

\*

\* All versions of Node Sass currently pass imports to importers before

\* loading them relative to the file in which the `@import` appears. This

\* behavior is considered incorrect and should not be relied on because it

\* violates the principle of \*locality\*, which says that it should be possible

\* to reason about a stylesheet without knowing everything about how the

\* entire system is set up. If a user tries to import a stylesheet relative to

\* another stylesheet, that import should \*always\* work. It shouldn’t be

\* possible for some configuration somewhere else to break it.

\*/

importer?: LegacyImporter<sync> | LegacyImporter<sync>[];

/\*\*

\* Additional built-in Sass functions that are available in all stylesheets.

\* This option takes an object whose keys are Sass function signatures and

\* whose values are [[LegacyFunction]]s. Each function should take the same

\* arguments as its signature.

\*

\* Functions are passed JavaScript representations of [Sass value

\* types](https://sass-lang.com/documentation/js-api#value-types), and must

\* return the same.

\*

\* \*\*Heads up!\*\* When writing custom functions, it’s important to ensure that

\* all the arguments are the types you expect. Otherwise, users’ stylesheets

\* could crash in hard-to-debug ways or, worse, compile to meaningless CSS.

\*

\* @example

\*

\* ```js

\* sass.render({

\* data: `

\* h1 {

\* font-size: pow(2, 5) \* 1px;

\* }`,

\* functions: {

\* // This function uses the synchronous API, and can be passed to either

\* // renderSync() or render().

\* 'pow($base, $exponent)': function(base, exponent) {

\* if (!(base instanceof sass.types.Number)) {

\* throw "$base: Expected a number.";

\* } else if (base.getUnit()) {

\* throw "$base: Expected a unitless number.";

\* }

\*

\* if (!(exponent instanceof sass.types.Number)) {

\* throw "$exponent: Expected a number.";

\* } else if (exponent.getUnit()) {

\* throw "$exponent: Expected a unitless number.";

\* }

\*

\* return new sass.types.Number(

\* Math.pow(base.getValue(), exponent.getValue()));

\* },

\*

\* // This function uses the asynchronous API, and can only be passed to

\* // render().

\* 'sqrt($number)': function(number, done) {

\* if (!(number instanceof sass.types.Number)) {

\* throw "$number: Expected a number.";

\* } else if (number.getUnit()) {

\* throw "$number: Expected a unitless number.";

\* }

\*

\* done(new sass.types.Number(Math.sqrt(number.getValue())));

\* }

\* }

\* }, function(err, result) {

\* console.log(result.css.toString());

\* // h1 {

\* // font-size: 32px;

\* // }

\* });

\* ```

\*

\* @category Plugins

\*/

functions?: {[key: string]: LegacyFunction<sync>};

/\*\*

\* By default, if the CSS document contains non-ASCII characters, Sass adds a

\* `@charset` declaration (in expanded output mode) or a byte-order mark (in

\* compressed mode) to indicate its encoding to browsers or other consumers.

\* If `charset` is `false`, these annotations are omitted.

\*

\* @category Output

\* @compatibility dart: "1.39.0", node: false

\*/

charset?: boolean;

/\*\*

\* If this option is set to `true`, Sass won’t print warnings that are caused

\* by dependencies. A “dependency” is defined as any file that’s loaded

\* through [[loadPaths]] or [[importer]]. Stylesheets that are imported

\* relative to the entrypoint are not considered dependencies.

\*

\* This is useful for silencing deprecation warnings that you can’t fix on

\* your own. However, please <em>also</em> notify your dependencies of the deprecations

\* so that they can get fixed as soon as possible!

\*

\* \*\*Heads up!\*\* If [[render]] or [[renderSync]] is called without

\* [[LegacyFileOptions.file]] or [[LegacyStringOptions.file]], <em>all</em>

\* stylesheets it loads will be considered dependencies. Since it doesn’t have

\* a path of its own, everything it loads is coming from a load path rather

\* than a relative import.

\*

\* @defaultValue `false`

\* @category Messages

\* @compatibility dart: "1.35.0", node: false

\*/

quietDeps?: boolean;

/\*\*

\* By default, Dart Sass will print only five instances of the same

\* deprecation warning per compilation to avoid deluging users in console

\* noise. If you set `verbose` to `true`, it will instead print every

\* deprecation warning it encounters.

\*

\* @defaultValue `false`

\* @category Messages

\* @compatibility dart: "1.35.0", node: false

\*/

verbose?: boolean;

/\*\*

\* An object to use to handle warnings and/or debug messages from Sass.

\*

\* By default, Sass emits warnings and debug messages to standard error, but

\* if [[Logger.warn]] or [[Logger.debug]] is set, this will invoke them

\* instead.

\*

\* The special value [[Logger.silent]] can be used to easily silence all

\* messages.

\*

\* @category Messages

\* @compatibility dart: "1.43.0", node: false

\*/

logger?: Logger;

}

/\*\*

\* If [[file]] is passed without [[data]], Sass will load the stylesheet at

\* [[file]] and compile it to CSS.

\*

\* @typeParam sync - This lets the TypeScript checker verify that

\* [[LegacyAsyncImporter]]s and [[LegacyAsyncFunction]]s aren't passed to

\* [[renderSync]].

\*/

export interface LegacyFileOptions<sync extends 'sync' | 'async'>

extends LegacySharedOptions<sync> {

/\*\*

\* The path to the file for Sass to load and compile. If the file’s extension

\* is `.scss`, it will be parsed as SCSS; if it’s `.sass`, it will be parsed

\* as the indented syntax; and if it’s `.css`, it will be parsed as plain CSS.

\* If it has no extension, it will be parsed as SCSS.

\*

\* @example

\*

\* ```js

\* sass.renderSync({file: "style.scss"});

\* ```

\*

\* @category Input

\* @compatibility feature: "Plain CSS files", dart: "1.11.0", node: "partial"

\*

\* Node Sass and older versions of Dart Sass support loading files with the

\* extension `.css`, but contrary to the specification they’re treated as SCSS

\* files rather than being parsed as CSS. This behavior has been deprecated

\* and should not be relied on. Any files that use Sass features should use

\* the `.scss` extension.

\*

\* All versions of Node Sass and Dart Sass otherwise support the file option

\* as described below.

\*/

file: string;

/\*\*

\* See [[LegacyStringOptions.file]] for documentation of passing [[file]] along

\* with [[data]].

\*

\* @category Input

\*/

data?: never;

}

/\*\*

\* If [[data]] is passed, Sass will use it as the contents of the stylesheet to

\* compile.

\*

\* @typeParam sync - This lets the TypeScript checker verify that

\* [[LegacyAsyncImporter]]s and [[LegacyAsyncFunction]]s aren't passed to

\* [[renderSync]].

\*

\* @category Legacy

\* @deprecated This only works with the legacy [[render]] and [[renderSync]]

\* APIs. Use [[StringOptions]] with [[compile]], [[compileString]],

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

\*/

export interface LegacyStringOptions<sync extends 'sync' | 'async'>

extends LegacySharedOptions<sync> {

/\*\*

\* The contents of the stylesheet to compile. Unless [[file]] is passed as

\* well, the stylesheet’s URL is set to `"stdin"`.

\*

\* By default, this stylesheet is parsed as SCSS. This can be controlled using

\* [[indentedSyntax]].

\*

\* @example

\*

\* ```js

\* sass.renderSync({

\* data: `

\* h1 {

\* font-size: 40px;

\* }`

\* });

\* ```

\*

\* @category Input

\*/

data: string;

/\*\*

\* If `file` and [[data]] are both passed, `file` is used as the path of the

\* stylesheet for error reporting, but [[data]] is used as the contents of the

\* stylesheet. In this case, `file`’s extension is not used to determine the

\* syntax of the stylesheet.

\*

\* @category Input

\*/

file?: string;

/\*\*

\* This flag controls whether [[data]] is parsed as the indented syntax or

\* not.

\*

\* @example

\*

\* ```js

\* sass.renderSync({

\* data: `

\* h1

\* font-size: 40px`,

\* indentedSyntax: true

\* });

\* ```

\*

\* @defaultValue `false`

\* @category Input

\*/

indentedSyntax?: boolean;

}

/\*\*

\* Options for [[render]] and [[renderSync]]. This can either be

\* [[LegacyFileOptions]] to load a file from disk, or [[LegacyStringOptions]] to

\* compile a string of Sass code.

\*

\* See [[LegacySharedOptions]] for options that are shared across both file and

\* string inputs.

\*

\* @category Legacy

\* @deprecated This only works with the legacy [[render]] and [[renderSync]]

\* APIs. Use [[Options]] with [[compile]], [[compileString]], [[compileAsync]],

\* and [[compileStringAsync]] instead.

\*/

export type LegacyOptions<sync extends 'sync' | 'async'> =

| LegacyFileOptions<sync>

| LegacyStringOptions<sync>;