import {LegacyPluginThis} from './plugin\_this';

/\*\*

\* A synchronous callback that implements a custom Sass function. This can be

\* passed to [[LegacySharedOptions.functions]] for either [[render]] or

\* [[renderSync]].

\*

\* If this throws an error, Sass will treat that as the function failing with

\* that error message.

\*

\* ```js

\* const result = sass.renderSync({

\* file: 'style.scss',

\* functions: {

\* "sum($arg1, $arg2)": (arg1, arg2) => {

\* if (!(arg1 instanceof sass.types.Number)) {

\* throw new Error("$arg1: Expected a number");

\* } else if (!(arg2 instanceof sass.types.Number)) {

\* throw new Error("$arg2: Expected a number");

\* }

\* return new sass.types.Number(arg1.getValue() + arg2.getValue());

\* }

\* }

\* });

\* ```

\*

\* @param args - One argument for each argument that's declared in the signature

\* that's passed to [[LegacySharedOptions.functions]]. If the signature [takes

\* arbitrary arguments](https://sass-lang.com/documentation/at-rules/function#taking-arbitrary-arguments),

\* they're passed as a single argument list in the last argument.

\*

\* @category Legacy

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

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

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

\*/

export type LegacySyncFunction = (

this: LegacyPluginThis,

...args: LegacyValue[]

) => LegacyValue;

/\*\*

\* An asynchronous callback that implements a custom Sass function. This can be

\* passed to [[LegacySharedOptions.functions]], but only for [[render]].

\*

\* An asynchronous function must return `undefined`. Its final argument will

\* always be a callback, which it should call with the result of the function

\* once it's done running.

\*

\* If this throws an error, Sass will treat that as the function failing with

\* that error message.

\*

\* ```js

\* sass.render({

\* file: 'style.scss',

\* functions: {

\* "sum($arg1, $arg2)": (arg1, arg2, done) => {

\* if (!(arg1 instanceof sass.types.Number)) {

\* throw new Error("$arg1: Expected a number");

\* } else if (!(arg2 instanceof sass.types.Number)) {

\* throw new Error("$arg2: Expected a number");

\* }

\* done(new sass.types.Number(arg1.getValue() + arg2.getValue()));

\* }

\* }

\* }, (result, error) => {

\* // ...

\* });

\* ```

\*

\* This is passed one argument for each argument that's declared in the

\* signature that's passed to [[LegacySharedOptions.functions]]. If the

\* signature [takes arbitrary

\* arguments](https://sass-lang.com/documentation/at-rules/function#taking-arbitrary-arguments),

\* they're passed as a single argument list in the last argument before the

\* callback.

\*

\* @category Legacy

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

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

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

\*/

export type LegacyAsyncFunction =

| ((this: LegacyPluginThis, done: (result: LegacyValue) => void) => void)

| ((

this: LegacyPluginThis,

arg1: LegacyValue,

done: LegacyAsyncFunctionDone

) => void)

| ((

this: LegacyPluginThis,

arg1: LegacyValue,

arg2: LegacyValue,

done: LegacyAsyncFunctionDone

) => void)

| ((

this: LegacyPluginThis,

arg1: LegacyValue,

arg2: LegacyValue,

arg3: LegacyValue,

done: LegacyAsyncFunctionDone

) => void)

| ((

this: LegacyPluginThis,

arg1: LegacyValue,

arg2: LegacyValue,

arg3: LegacyValue,

arg4: LegacyValue,

done: LegacyAsyncFunctionDone

) => void)

| ((

this: LegacyPluginThis,

arg1: LegacyValue,

arg2: LegacyValue,

arg3: LegacyValue,

arg4: LegacyValue,

arg5: LegacyValue,

done: LegacyAsyncFunctionDone

) => void)

| ((

this: LegacyPluginThis,

arg1: LegacyValue,

arg2: LegacyValue,

arg3: LegacyValue,

arg4: LegacyValue,

arg5: LegacyValue,

arg6: LegacyValue,

done: LegacyAsyncFunctionDone

) => void)

| ((

this: LegacyPluginThis,

...args: [...LegacyValue[], LegacyAsyncFunctionDone]

) => void);

/\*\*

\* The function called by a [[LegacyAsyncFunction]] to indicate that it's

\* finished.

\*

\* @param result - If this is a [[LegacyValue]], that indicates that the

\* function call completed successfully. If it's a [[types.Error]], that

\* indicates that the function call failed.

\*

\* @category Legacy

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

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

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

\*/

export type LegacyAsyncFunctionDone = (

result: LegacyValue | types.Error

) => void;

/\*\*

\* A callback that implements a custom Sass function. For [[renderSync]], this

\* must be a [[LegacySyncFunction]] which returns its result directly; for

\* [[render]], it may be either a [[LegacySyncFunction]] or a

\* [[LegacyAsyncFunction]] which calls a callback with its result.

\*

\* See [[LegacySharedOptions.functions]] for more details.

\*

\* @category Legacy

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

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

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

\*/

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

? LegacySyncFunction | LegacyAsyncFunction

: LegacySyncFunction;

/\*\*

\* A type representing all the possible values that may be passed to or returned

\* from a [[LegacyFunction]].

\*

\* @category Legacy

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

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

\* and [[compileStringAsync]] instead.

\*/

export type LegacyValue =

| types.Null

| types.Number

| types.String

| types.Boolean

| types.Color

| types.List

| types.Map;

/\*\*

\* A shorthand for `sass.types.Boolean.TRUE`.

\*

\* @category Legacy

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

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

\* and [[compileStringAsync]] instead.

\*/

export const TRUE: types.Boolean<true>;

/\*\*

\* A shorthand for `sass.types.Boolean.FALSE`.

\*

\* @category Legacy

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

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

\* and [[compileStringAsync]] instead.

\*/

export const FALSE: types.Boolean<false>;

/\*\*

\* A shorthand for `sass.types.Null.NULL`.

\*

\* @category Legacy

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

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

\* and [[compileStringAsync]] instead.

\*/

export const NULL: types.Null;

/\*\*

\* The namespace for value types used in the legacy function API.

\*

\* @category Legacy

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

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

\* and [[compileStringAsync]] instead.

\*/

export namespace types {

/\*\*

\* The class for Sass's singleton [`null`

\* value](https://sass-lang.com/documentation/values/null). The value itself

\* can be accessed through the [[NULL]] field.

\*/

export class Null {

/\*\* Sass's singleton `null` value. \*/

static readonly NULL: Null;

}

/\*\*

\* Sass's [number type](https://sass-lang.com/documentation/values/numbers).

\*/

export class Number {

/\*\*

\* @param value - The numeric value of the number.

\*

\* @param unit - If passed, the number's unit.

\*

\* Complex units can be represented as

\* `<unit>\*<unit>\*.../<unit>\*<unit>\*...`, with numerator units on the

\* left-hand side of the `/` and denominator units on the right. A number

\* with only numerator units may omit the `/` and the units after it, and a

\* number with only denominator units may be represented

\* with no units before the `/`.

\*

\* @example

\*

\* ```scss

\* new sass.types.Number(0.5); // == 0.5

\* new sass.types.Number(10, "px"); // == 10px

\* new sass.types.Number(10, "px\*px"); // == 10px \* 1px

\* new sass.types.Number(10, "px/s"); // == math.div(10px, 1s)

\* new sass.types.Number(10, "px\*px/s\*s"); // == 10px \* math.div(math.div(1px, 1s), 1s)

\* ```

\*/

constructor(value: number, unit?: string);

/\*\*

\* Returns the value of the number, ignoring units.

\*

\* \*\*Heads up!\*\* This means that `96px` and `1in` will return different

\* values, even though they represent the same length.

\*

\* @example

\*

\* ```js

\* const number = new sass.types.Number(10, "px");

\* number.getValue(); // 10

\* ```

\*/

getValue(): number;

/\*\*

\* Destructively modifies this number by setting its numeric value to

\* `value`, independent of its units.

\*

\* @deprecated Use [[constructor]] instead.

\*/

setValue(value: number): void;

/\*\*

\* Returns a string representation of this number's units. Complex units are

\* returned in the same format that [[constructor]] accepts them.

\*

\* @example

\*

\* ```js

\* // number is `10px`.

\* number.getUnit(); // "px"

\*

\* // number is `math.div(10px, 1s)`.

\* number.getUnit(); // "px/s"

\* ```

\*/

getUnit(): string;

/\*\*

\* Destructively modifies this number by setting its units to `unit`,

\* independent of its numeric value. Complex units are specified in the same

\* format as [[constructor]].

\*

\* @deprecated Use [[constructor]] instead.

\*/

setUnit(unit: string): void;

}

/\*\*

\* Sass's [string type](https://sass-lang.com/documentation/values/strings).

\*

\* \*\*Heads up!\*\* This API currently provides no way of distinguishing between

\* a [quoted](https://sass-lang.com/documentation/values/strings#quoted) and

\* [unquoted](https://sass-lang.com/documentation/values/strings#unquoted)

\* string.

\*/

export class String {

/\*\*

\* Creates an unquoted string with the given contents.

\*

\* \*\*Heads up!\*\* This API currently provides no way of creating a

\* [quoted](https://sass-lang.com/documentation/values/strings#quoted)

\* string.

\*/

constructor(value: string);

/\*\*

\* Returns the contents of the string. If the string contains escapes,

\* those escapes are included literally if it’s

\* [unquoted](https://sass-lang.com/documentation/values/strings#unquoted),

\* while the values of the escapes are included if it’s

\* [quoted](https://sass-lang.com/documentation/values/strings#quoted).

\*

\* @example

\*

\* ```

\* // string is `Arial`.

\* string.getValue(); // "Arial"

\*

\* // string is `"Helvetica Neue"`.

\* string.getValue(); // "Helvetica Neue"

\*

\* // string is `\1F46D`.

\* string.getValue(); // "\\1F46D"

\*

\* // string is `"\1F46D"`.

\* string.getValue(); // "👭"

\* ```

\*/

getValue(): string;

/\*\*

\* Destructively modifies this string by setting its numeric value to

\* `value`.

\*

\* \*\*Heads up!\*\* Even if the string was originally quoted, this will cause

\* it to become unquoted.

\*

\* @deprecated Use [[constructor]] instead.

\*/

setValue(value: string): void;

}

/\*\*

\* Sass's [boolean type](https://sass-lang.com/documentation/values/booleans).

\*

\* Custom functions should respect Sass’s notion of

\* [truthiness](https://sass-lang.com/documentation/at-rules/control/if#truthiness-and-falsiness)

\* by treating `false` and `null` as falsey and everything else as truthy.

\*

\* \*\*Heads up!\*\* Boolean values can't be constructed, they can only be

\* accessed through the [[TRUE]] and [[FALSE]] constants.

\*/

export class Boolean<T extends boolean = boolean> {

/\*\*

\* Returns `true` if this is Sass's `true` value and `false` if this is

\* Sass's `false` value.

\*

\* @example

\*

\* ```js

\* // boolean is `true`.

\* boolean.getValue(); // true

\* boolean === sass.types.Boolean.TRUE; // true

\*

\* // boolean is `false`.

\* boolean.getValue(); // false

\* boolean === sass.types.Boolean.FALSE; // true

\* ```

\*/

getValue(): T;

/\*\* Sass's `true` value. \*/

static readonly TRUE: Boolean<true>;

/\*\* Sass's `false` value. \*/

static readonly FALSE: Boolean<false>;

}

/\*\*

\* Sass's [color type](https://sass-lang.com/documentation/values/colors).

\*/

export class Color {

/\*\*

\* Creates a new Sass color with the given red, green, blue, and alpha

\* channels. The red, green, and blue channels must be integers between 0

\* and 255 (inclusive), and alpha must be between 0 and 1 (inclusive).

\*

\* @example

\*

\* ```js

\* new sass.types.Color(107, 113, 127); // #6b717f

\* new sass.types.Color(0, 0, 0, 0); // rgba(0, 0, 0, 0)

\* ```

\*/

constructor(r: number, g: number, b: number, a?: number);

/\*\*

\* Creates a new Sass color with alpha, red, green, and blue channels taken

\* from respective two-byte chunks of a hexidecimal number.

\*

\* @example

\*

\* ```js

\* new sass.types.Color(0xff6b717f); // #6b717f

\* new sass.types.Color(0x00000000); // rgba(0, 0, 0, 0)

\* ```

\*/

constructor(argb: number);

/\*\*

\* Returns the red channel of the color as an integer from 0 to 255.

\*

\* @example

\*

\* ```js

\* // color is `#6b717f`.

\* color.getR(); // 107

\*

\* // color is `#b37399`.

\* color.getR(); // 179

\* ```

\*/

getR(): number;

/\*\*

\* Sets the red channel of the color. The value must be an integer between 0

\* and 255 (inclusive).

\*

\* @deprecated Use [[constructor]] instead.

\*/

setR(value: number): void;

/\*\*

\* Returns the green channel of the color as an integer from 0 to 255.

\*

\* @example

\*

\* ```js

\* // color is `#6b717f`.

\* color.getG(); // 113

\*

\* // color is `#b37399`.

\* color.getG(); // 115

\* ```

\*/

getG(): number;

/\*\*

\* Sets the green channel of the color. The value must be an integer between

\* 0 and 255 (inclusive).

\*

\* @deprecated Use [[constructor]] instead.

\*/

setG(value: number): void;

/\*\*

\* Returns the blue channel of the color as an integer from 0 to 255.

\*

\* @example

\*

\* ```js

\* // color is `#6b717f`.

\* color.getB(); // 127

\*

\* // color is `#b37399`.

\* color.getB(); // 153

\* ```

\*/

getB(): number;

/\*\*

\* Sets the blue channel of the color. The value must be an integer between

\* 0 and 255 (inclusive).

\*

\* @deprecated Use [[constructor]] instead.

\*/

setB(value: number): void;

/\*\*

\* Returns the alpha channel of the color as a number from 0 to 1.

\*

\* @example

\*

\* ```js

\* // color is `#6b717f`.

\* color.getA(); // 1

\*

\* // color is `transparent`.

\* color.getA(); // 0

\* ```

\*/

getA(): number;

/\*\*

\* Sets the alpha channel of the color. The value must be between 0 and 1

\* (inclusive).

\*

\* @deprecated Use [[constructor]] instead.

\*/

setA(value: number): void;

}

/\*\*

\* Sass's [list type](https://sass-lang.com/documentation/values/lists).

\*

\* \*\*Heads up!\*\* This list type’s methods use 0-based indexing, even though

\* within Sass lists use 1-based indexing. These methods also don’t support

\* using negative numbers to index backwards from the end of the list.

\*/

export class List {

/\*\*

\* Creates a new Sass list.

\*

\* \*\*Heads up!\*\* The initial values of the list elements are undefined.

\* These elements must be set using [[setValue]] before accessing them or

\* passing the list back to Sass.

\*

\* @example

\*

\* ```js

\* const list = new sass.types.List(3);

\* list.setValue(0, new sass.types.Number(10, "px"));

\* list.setValue(1, new sass.types.Number(15, "px"));

\* list.setValue(2, new sass.types.Number(32, "px"));

\* list; // 10px, 15px, 32px

\* ```

\*

\* @param length - The number of (initially undefined) elements in the list.

\* @param commaSeparator - If `true`, the list is comma-separated; otherwise,

\* it's space-separated. Defaults to `true`.

\*/

constructor(length: number, commaSeparator?: boolean);

/\*\*

\* Returns the element at `index`, or `undefined` if that value hasn't yet

\* been set.

\*

\* @example

\*

\* ```js

\* // list is `10px, 15px, 32px`

\* list.getValue(0); // 10px

\* list.getValue(2); // 32px

\* ```

\*

\* @param index - A (0-based) index into this list.

\* @throws `Error` if `index` is less than 0 or greater than or equal to the

\* number of elements in this list.

\*/

getValue(index: number): LegacyValue | undefined;

/\*\*

\* Sets the element at `index` to `value`.

\*

\* @example

\*

\* ```js

\* // list is `10px, 15px, 32px`

\* list.setValue(1, new sass.types.Number(18, "px"));

\* list; // 10px, 18px, 32px

\* ```

\*

\* @param index - A (0-based) index into this list.

\* @throws `Error` if `index` is less than 0 or greater than or equal to the

\* number of elements in this list.

\*/

setValue(index: number, value: LegacyValue): void;

/\*\*

\* Returns `true` if this list is comma-separated and `false` otherwise.

\*

\* @example

\*

\* ```js

\* // list is `10px, 15px, 32px`

\* list.getSeparator(); // true

\*

\* // list is `1px solid`

\* list.getSeparator(); // false

\* ```

\*/

getSeparator(): boolean;

/\*\*

\* Sets whether the list is comma-separated.

\*

\* @param isComma - `true` to make the list comma-separated, `false` otherwise.

\*/

setSeparator(isComma: boolean): void;

/\*\*

\* Returns the number of elements in the list.

\*

\* @example

\*

\* ```js

\* // list is `10px, 15px, 32px`

\* list.getLength(); // 3

\*

\* // list is `1px solid`

\* list.getLength(); // 2

\* ```

\*/

getLength(): number;

}

/\*\*

\* Sass's [map type](https://sass-lang.com/documentation/values/maps).

\*

\* \*\*Heads up!\*\* This map type is represented as a list of key-value pairs

\* rather than a mapping from keys to values. The only way to find the value

\* associated with a given key is to iterate through the map checking for that

\* key. Maps created through this API are still forbidden from having duplicate

\* keys.

\*/

export class Map {

/\*\*

\* Creates a new Sass map.

\*

\* \*\*Heads up!\*\* The initial keys and values of the map are undefined. They

\* must be set using [[setKey]] and [[setValue]] before accessing them or

\* passing the map back to Sass.

\*

\* @example

\*

\* ```js

\* const map = new sass.types.Map(2);

\* map.setKey(0, new sass.types.String("width"));

\* map.setValue(0, new sass.types.Number(300, "px"));

\* map.setKey(1, new sass.types.String("height"));

\* map.setValue(1, new sass.types.Number(100, "px"));

\* map; // (width: 300px, height: 100px)

\* ```

\*

\* @param length - The number of (initially undefined) key/value pairs in the map.

\*/

constructor(length: number);

/\*\*

\* Returns the value in the key/value pair at `index`.

\*

\* @example

\*

\* ```js

\* // map is `(width: 300px, height: 100px)`

\* map.getValue(0); // 300px

\* map.getValue(1); // 100px

\* ```

\*

\* @param index - A (0-based) index of a key/value pair in this map.

\* @throws `Error` if `index` is less than 0 or greater than or equal to the

\* number of pairs in this map.

\*/

getValue(index: number): LegacyValue;

/\*\*

\* Sets the value in the key/value pair at `index` to `value`.

\*

\* @example

\*

\* ```js

\* // map is `("light": 200, "medium": 400, "bold": 600)`

\* map.setValue(1, new sass.types.Number(300));

\* map; // ("light": 200, "medium": 300, "bold": 600)

\* ```

\*

\* @param index - A (0-based) index of a key/value pair in this map.

\* @throws `Error` if `index` is less than 0 or greater than or equal to the

\* number of pairs in this map.

\*/

setValue(index: number, value: LegacyValue): void;

/\*\*

\* Returns the key in the key/value pair at `index`.

\*

\* @example

\*

\* ```js

\* // map is `(width: 300px, height: 100px)`

\* map.getKey(0); // width

\* map.getKey(1); // height

\* ```

\*

\* @param index - A (0-based) index of a key/value pair in this map.

\* @throws `Error` if `index` is less than 0 or greater than or equal to the

\* number of pairs in this map.

\*/

getKey(index: number): LegacyValue;

/\*\*

\* Sets the value in the key/value pair at `index` to `value`.

\*

\* @example

\*

\* ```js

\* // map is `("light": 200, "medium": 400, "bold": 600)`

\* map.setValue(1, new sass.types.String("lighter"));

\* map; // ("lighter": 200, "medium": 300, "bold": 600)

\* ```

\*

\* @param index - A (0-based) index of a key/value pair in this map.

\* @throws `Error` if `index` is less than 0 or greater than or equal to the

\* number of pairs in this map.

\*/

setKey(index: number, key: LegacyValue): void;

/\*\*

\* Returns the number of key/value pairs in this map.

\*

\* @example

\*

\* ```js

\* // map is `("light": 200, "medium": 400, "bold": 600)`

\* map.getLength(); // 3

\*

\* // map is `(width: 300px, height: 100px)`

\* map.getLength(); // 2

\* ```

\*/

getLength(): number;

}

/\*\*

\* An error that can be returned from a Sass function to signal that it

\* encountered an error. This is the only way to signal an error

\* asynchronously from a [[LegacyAsyncFunction]].

\*/

export class Error {

constructor(message: string);

}

}