/\*\*

\* The `diagnostics\_channel` module provides an API to create named channels

\* to report arbitrary message data for diagnostics purposes.

\*

\* It can be accessed using:

\*

\* ```js

\* import diagnostics\_channel from 'diagnostics\_channel';

\* ```

\*

\* It is intended that a module writer wanting to report diagnostics messages

\* will create one or many top-level channels to report messages through.

\* Channels may also be acquired at runtime but it is not encouraged

\* due to the additional overhead of doing so. Channels may be exported for

\* convenience, but as long as the name is known it can be acquired anywhere.

\*

\* If you intend for your module to produce diagnostics data for others to

\* consume it is recommended that you include documentation of what named

\* channels are used along with the shape of the message data. Channel names

\* should generally include the module name to avoid collisions with data from

\* other modules.

\* @experimental

\* @see [source](https://github.com/nodejs/node/blob/v17.0.0/lib/diagnostics\_channel.js)

\*/

declare module 'diagnostics\_channel' {

/\*\*

\* Check if there are active subscribers to the named channel. This is helpful if

\* the message you want to send might be expensive to prepare.

\*

\* This API is optional but helpful when trying to publish messages from very

\* performance-sensitive code.

\*

\* ```js

\* import diagnostics\_channel from 'diagnostics\_channel';

\*

\* if (diagnostics\_channel.hasSubscribers('my-channel')) {

\* // There are subscribers, prepare and publish message

\* }

\* ```

\* @since v15.1.0, v14.17.0

\* @param name The channel name

\* @return If there are active subscribers

\*/

function hasSubscribers(name: string): boolean;

/\*\*

\* This is the primary entry-point for anyone wanting to interact with a named

\* channel. It produces a channel object which is optimized to reduce overhead at

\* publish time as much as possible.

\*

\* ```js

\* import diagnostics\_channel from 'diagnostics\_channel';

\*

\* const channel = diagnostics\_channel.channel('my-channel');

\* ```

\* @since v15.1.0, v14.17.0

\* @param name The channel name

\* @return The named channel object

\*/

function channel(name: string): Channel;

type ChannelListener = (name: string, message: unknown) => void;

/\*\*

\* The class `Channel` represents an individual named channel within the data

\* pipeline. It is use to track subscribers and to publish messages when there

\* are subscribers present. It exists as a separate object to avoid channel

\* lookups at publish time, enabling very fast publish speeds and allowing

\* for heavy use while incurring very minimal cost. Channels are created with {@link channel}, constructing a channel directly

\* with `new Channel(name)` is not supported.

\* @since v15.1.0, v14.17.0

\*/

class Channel {

readonly name: string;

/\*\*

\* Check if there are active subscribers to this channel. This is helpful if

\* the message you want to send might be expensive to prepare.

\*

\* This API is optional but helpful when trying to publish messages from very

\* performance-sensitive code.

\*

\* ```js

\* import diagnostics\_channel from 'diagnostics\_channel';

\*

\* const channel = diagnostics\_channel.channel('my-channel');

\*

\* if (channel.hasSubscribers) {

\* // There are subscribers, prepare and publish message

\* }

\* ```

\* @since v15.1.0, v14.17.0

\*/

readonly hasSubscribers: boolean;

private constructor(name: string);

/\*\*

\* Register a message handler to subscribe to this channel. This message handler

\* will be run synchronously whenever a message is published to the channel. Any

\* errors thrown in the message handler will trigger an `'uncaughtException'`.

\*

\* ```js

\* import diagnostics\_channel from 'diagnostics\_channel';

\*

\* const channel = diagnostics\_channel.channel('my-channel');

\*

\* channel.subscribe((message, name) => {

\* // Received data

\* });

\* ```

\* @since v15.1.0, v14.17.0

\* @param onMessage The handler to receive channel messages

\*/

subscribe(onMessage: ChannelListener): void;

/\*\*

\* Remove a message handler previously registered to this channel with `channel.subscribe(onMessage)`.

\*

\* ```js

\* import diagnostics\_channel from 'diagnostics\_channel';

\*

\* const channel = diagnostics\_channel.channel('my-channel');

\*

\* function onMessage(message, name) {

\* // Received data

\* }

\*

\* channel.subscribe(onMessage);

\*

\* channel.unsubscribe(onMessage);

\* ```

\* @since v15.1.0, v14.17.0

\* @param onMessage The previous subscribed handler to remove

\*/

unsubscribe(onMessage: ChannelListener): void;

}

}

declare module 'node:diagnostics\_channel' {

export \* from 'diagnostics\_channel';

}