Diagnostics Channel Support

Stability: Experimental.

Undici supports the `diagnostics_channel` (currently available only on Node.js v16+). It is the preferred way to instrument Undici and retrieve internal information.

The channels available are the following.

`undici:request:create`

This message is published when a new outgoing request is created.

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:request:create').subscribe(({ request }) => {
  console.log('origin', request.origin)
  console.log('completed', request.completed)
  console.log('method', request.method)
  console.log('path', request.path)
  console.log('headers') // raw text, e.g: 'bar: bar\r\n'
  request.addHeader('hello', 'world')
  console.log('headers', request.headers) // e.g. 'bar: bar\r\nhello: world\r\n'
})
```

Note: a request is only loosely completed to a given socket.

`undici:request:bodySent`

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:request:bodySent').subscribe(({ request }) => {
  // request is the same object undici:request:create
})
```

`undici:request:headers`

This message is published after the response headers have been received, i.e. the response has been completed.

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:request:headers').subscribe(({ request, response
}) => {
  // request is the same object undici:request:create
  console.log('statusCode', response.statusCode)
  console.log(response.statusText)
  // response.headers are buffers.
```

```
  console.log(response.headers.map((x) => x.toString()))
})
```

`undici:request:trailers`

This message is published after the response body and trailers have been received, i.e. the response has been completed.

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:request:trailers').subscribe(({ request, trailers
}) => {
  // request is the same object undici:request:create
  console.log('completed', request.completed)
  // trailers are buffers.
  console.log(trailers.map((x) => x.toString()))
})
```

`undici:request:error`

This message is published if the request is going to error, but it has not errored yet.

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:request:error').subscribe(({ request, error }) =>
{
  // request is the same object undici:request:create
})
```

`undici:client:sendHeaders`

This message is published right before the first byte of the request is written to the socket.

Note: It will publish the exact headers that will be sent to the server in raw format.

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:client:sendHeaders').subscribe(({ request,
headers, socket }) => {
  // request is the same object undici:request:create
  console.log(`Full headers list ${headers.split('\r\n')}`);
})
```

`undici:client:beforeConnect`

This message is published before creating a new connection for **any** request. You can not assume that this event is related to any specific request.

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:client:beforeConnect').subscribe(({
connectParams, connector }) => {
  // const { host, hostname, protocol, port, servername } = connectParams
  // connector is a function that creates the socket
})
```

`undici:client:connected`

This message is published after a connection is established.

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:client:connected').subscribe(({ socket,
connectParams, connector }) => {
  // const { host, hostname, protocol, port, servername } = connectParams
 // connector is a function that creates the socket
})
```

`undici:client:connectError`

This message is published if it did not succeed to create new connection

```
import diagnosticsChannel from 'diagnostics_channel'

diagnosticsChannel.channel('undici:client:connectError').subscribe(({ error, socket,
connectParams, connector }) => {
  // const { host, hostname, protocol, port, servername } = connectParams
  // connector is a function that creates the socket
  console.log(`Connect failed with ${error.message}`)
})
```