Class

NetworkConnection

Connect to an endpoint on the network to send and receive data.

iOS 26.0+ | iPadOS 26.0+ | Mac Catalyst 26.0+ | macOS 26.0+ | tvOS 26.0+ | visionOS 26.0+ | watchOS 26.0+

```
final class NetworkConnection<ApplicationProtocol> where Application
Protocol : NetworkProtocolOptions
```

Overview

A connection handles establishment of any transport, security, and application-level protocols required to transmit and receive user data. Connections may make multiple establishment attempts before the connection is ready.

Topics

Initializers

`convenience init(to: NWEndpoint, using: () -> ApplicationProtocol)`

Create a new connection to an endpoint, with protocol stack.

`convenience init(to: any Connectable, using: () -> ApplicationProtocol)`

Create a new connection to an endpoint, with protocol stack.

`convenience init(to: NWEndpoint, using: NWParametersBuilder<Application Protocol>)`

```
convenience init(to: NWEndpoint, using: NWParametersBuilder<ApplicationProtocol>)
```

Create a new outbound connection to an endpoint, with parameters. The parameters determine the protocols to be used for the connection, and their options.

```
convenience init(to: any Connectable, using: NWParametersBuilder<ApplicationProtocol>)
```

Create a new outbound connection to an endpoint, with parameters. The parameters determine the protocols to be used for the connection, and their options.

```
convenience init(to: any Connectable, using: () -> ApplicationProtocol)
```

```
convenience init(to: NWEndpoint, using: () -> ApplicationProtocol)
```

```
convenience init(to: any Connectable, using: NWParametersBuilder<ApplicationProtocol>)
```

Instance Properties

```
var applicationError: NWProtocolQUIC.ApplicationError
```

The QUIC application error code to send for the connection, or received from the peer.

```
var currentPath: NWPath?
```

Current path for the connection, which can be used to extract interface and effective endpoint information

```
var datagrams: QUIC.Datagrams<QUICDatagram>
```

Access connection-wide unreliable datagrams over QUIC. Subsequent accesses to this object will return the same reference. All incoming datagrams for the entire QUIC connection will be received on this `SubConnection` once invoked.

```
var keepalive: NWProtocolQUIC.Metadata.KeepAliveBehavior
```

Set the QUIC connection keepalive interval.

```
var localEndpoint: NWEndpoint?
```

The local endpoint of the connection

```
var negotiatedALPN: String?
```

Return the negotiated application protocol used when establishing the connection

```
var remoteEndpoint: NWEndpoint?
```

The remote endpoint of the connection

`var remoteIdleTimeout: Int`

Access the idle_timeout value in milliseconds received from the peer in the transport parameters.

`var remoteMaxStreamsBidirectional: Int`

Get the maximum number of bidirectional streams advertised by peer that an application is allowed to create.

`var remoteMaxStreamsUnidirectional: Int`

Get the maximum number of unidirectional streams advertised by peer that an application is allowed to create.

`var securityProtocolMetadata: sec_protocol_metadata_t`

Access the sec_protocol_metadata_t for the QUIC Connection. See <Security/SecProtocolMetadata.h> for functions to further access security metadata.

`var usableDatagramFrameSize: Int`

Get the usable size of a datagram frame from a QUIC datagram flow.

Instance Methods

`func inboundStreams((QUIC.Stream<QUICStream>) async throws -> Void) async throws`

Handle inbound streams and provide a closure on which callback handlers will be executed. When the `NetworkConnection<QUIC>` state moves to `ready`, the internal listener is registered with the system and can receive incoming streams on the multiplexing instance. `inboundStreams` should only be called once on a `NetworkConnection<QUIC>`, and multiple calls to run will throw an exception.

`func inboundStreams<NewApplicationProtocol>(prepending: (QUICStream) -> NewApplicationProtocol, (QUIC.Stream<NewApplicationProtocol>) async throws -> Void) async throws`

Handle inbound streams and provide a closure on which callback handlers will be executed. When the `NetworkConnection<QUIC>` state moves to `ready`, the internal listener is registered with the system and can receive incoming streams on the multiplexing instance. `inboundStreams` should only be called once on a `NetworkConnection<QUIC>`, and multiple calls to run will throw an exception.

`func onBetterPathUpdate((NetworkConnection<ApplicationProtocol>, Bool) -> Void) -> Self`

A better path being available indicates that the system thinks there is a preferred path or interface to use, compared to the one this connection is actively using. As an example, the

connection is established over an expensive cellular interface and an unmetered Wi-Fi interface is now available.

```
func onPathUpdate((NetworkConnection<ApplicationProtocol>, NWPath) -> Void) -> Self
```

Set a closure to be called when the connection's path has changed, which may be called multiple times until the connection is cancelled.

```
func onStateUpdate((NetworkConnection<ApplicationProtocol>, NetworkChannel<ApplicationProtocol>.State) -> Void) -> Self
```

Set a closure to be called when the connection's state changes, which may be called multiple times until the connection is cancelled.

```
func onViabilityUpdate((NetworkConnection<ApplicationProtocol>, Bool) -> Void) -> Self
```

Set a closure to be called when the connection's viability changes, which may be called multiple times until the connection is cancelled.

```
func openStream(directionality: QUICStream.Directionality) async throws -> QUIC.Stream<QUICStream>
```

Initiate a new data stream over QUIC. When invoked with no parameters, the default stream type will be bidirectional. Unidirectional streams can be initiated by setting the optional `bidirectional` parameter to false.

```
func openStream<NewApplicationProtocol>(directionality: QUICStream.Directionality, (QUICStream) -> NewApplicationProtocol) async throws -> QUIC.Stream<NewApplicationProtocol>
```

Initiate a new data stream over QUIC. When invoked with no parameters, the default stream type will be bidirectional. Unidirectional streams can be initiated by setting the optional `bidirectional` parameter to false.

```
func start() -> Self
```

Initiate some action to open the connection on the network like making a handshake, initiating a multiplexing session, etc. Starts the connection, which will cause the connection to evaluate its path, do resolution, and try to become ready (connected). `NetworkConnection` establishment is asynchronous. `onStateUpdate` will be called when the state changes. If the connection cannot be established, the state will transition to `waiting` with an associated error describing the reason. If an unrecoverable error is encountered, the state will transition to `failed` with an associated error value. If the connection is established, the state will transition to `ready`.

```
func tryNextEndpoint()
```

Cancel the currently connected endpoint, causing the connection to fall through to the next endpoint if available, or to go to the waiting state if no more endpoints are available.

Relationships

Inherits From

`NetworkChannel`

Conforms To

`CustomDebugStringConvertible`
`Equatable`
`Hashable`
`Identifiable`
`Sendable`
`SendableMetatype`