

Framework

Network

Create network connections to send and receive data using transport and security protocols.

iOS 12.0+ | iPadOS 12.0+ | Mac Catalyst 13.0+ | macOS 10.14+ | tvOS 12.0+ | visionOS 1.0+ | watchOS 6.0+

Overview

Use this framework when you need direct access to protocols like TLS, TCP, and UDP for your custom application protocols. Continue to use [URLSession](#), which is built upon this framework, for loading HTTP- and URL-based resources. For in-depth advice on where to start with networking, see [TN3151: Choosing the right networking API](#).

Note

watchOS supports Network framework for specific use cases. For more details, see [TN3135: Low-level networking on watchOS](#).

Topics

Essentials

`enum` `NWEndpoint`

A local or remote endpoint in a network connection.

`class` `NWParameters`

An object that stores the protocols to use for connections, options for sending data, and network path constraints.

Connections and Listeners

`class NWConnection`

A bidirectional data connection between a local endpoint and a remote endpoint.

`class NWListener`

An object you use to listen for incoming network connections.

`class NWBrowser`

An object you use to browse for available network services.

`class NWConnectionGroup`

An object you use to communicate with a group of endpoints, such as an IP multicast group on a local network.

`class NWEthernetChannel`

An object you use to send and receive custom Ethernet frames.

Network Protocols

Configure protocol options to use with connections and listeners, and inspect the results of protocol handshakes.

`{}` Building a custom peer-to-peer protocol

Use networking frameworks to create a custom protocol for playing a game across iOS, iPadOS, watchOS, and tvOS devices.

`class NWProtocolTCP`

A network protocol for connections that use the Transmission Control Protocol.

`class NWProtocolTLS`

A network protocol for connections that use Transport Layer Security.

`class NWProtocolQUIC`

A network protocol for connections that use the QUIC transport protocol.

`class NWProtocolUDP`

A network protocol for connections that use the User Datagram Protocol.

`class NWProtocolIP`

A network protocol for configuring the Internet Protocol on connections.

`class NWProtocolWebSocket`

A network protocol for connections that use WebSocket.

`class NWProtocolFramer`

A customizable network protocol for defining application message parsers.

Network Security and Privacy

 Security Options

Configure security options for TLS handshakes.

 Privacy Management

Configure parameters related to user privacy.

 Creating an Identity for Local Network TLS

Learn how to create and use a digital identity in your application for local network TLS.

Paths and Interfaces

`struct NWPath`

An object that contains information about the properties of the network that a connection uses, or that are available to your app.

`class NWPathMonitor`

An observer that you use to monitor and react to network changes.

`struct NWInterface`

An interface that a network connection uses to send and receive data.

Errors

`enum NSError`

The errors returned by objects in the Network framework.

Network Debugging

 Choosing a Network Debugging Tool

Decide which tool works best for your network debugging problem.

 Debugging HTTP Server-Side Errors

Understand HTTP server-side errors and how to debug them.



Debugging HTTPS Problems with CFNetwork Diagnostic Logging

Use CFNetwork diagnostic logging to investigate HTTP and HTTPS problems.



Recording a Packet Trace

Learn how to record a low-level trace of network traffic.



Taking Advantage of Third-Party Network Debugging Tools

Learn about the available third-party network debugging tools.



Testing and Debugging L4S in Your App

Learn how to verify your app on an L4S-capable host and network to improve your app's responsiveness.

C-Language Symbols

Access Network framework symbols used in C.



C-Language Symbols

Structures

`struct nw_interface_radio_type_t`

`struct nw_multipath_version_t`

`struct nw_path_unsatisfied_reason_t`

`struct nw_quic_stream_type_t`

`struct Bonjour`

A browser that discovers Bonjour services.

`struct BonjourListenerProvider`

Advertise a Bonjour service.

`struct Coder`

A protocol that frames and encodes/decodes Codable types.

`struct DefaultProtocolStorage`

`struct Framer`

An instance of a Framer protocol to load into a protocol stack.

`struct IP`

The system definition of the Internet Protocol (IP).

`struct NWParametersBuilder`

An opaque class that is responsible for creating and configuring NWParameters based on the parameterized protocol stack.

`struct NWTXTRecord`

A dictionary representing a TXT record in a DNS packet.

`struct NetworkJSONCoder`

`struct NetworkPropertyListCoder`

`struct ProtocolMetadataBuilder`

A resultBuilder for configuring metadata in send methods in a declarative way.

`struct ProtocolStackBuilder`

A resultBuilder for specifying and configuring protocol stacks in a declarative way

`struct ProxyConfiguration`

A proxy configuration for Relays, Oblivious HTTP, HTTP CONNECT, or SOCKSv5.

`struct QUIC`

The system definition of the QUIC protocol.

`struct QUICDatagram`

Send and receive unreliable datagrams over QUIC via RFC 9221

`struct QUICStream`

A QUIC stream that runs over a QUIC connection.

`struct TCP`

The system definition of the Transmission Control Protocol (TCP).

`struct TLS`

The system definition of the Transport Layer Security (TLS) protocol.

`struct TLV`

A Type-Length-Value (TLV) framing protocol.

`struct TXTRecordDecoder`

`struct UDP`

The system definition of the User Datagram Protocol (UDP).

```
struct UnexpectedEndpointType
```

An error generated when an unexpected endpoint type is supplied.

```
struct WebSocket
```

The system definition of the WebSocket protocol.

```
struct nw_link_quality_t
```

Classes

```
class NWMultiplexGroup
```

Discover advertised services and devices on the network.

```
class NetworkChannel
```

A base class supporting sending and receiving data through an arbitrary network channel.

```
class NetworkConnection
```

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

```
class NetworkListener
```

Listen for incoming network connections.

Reference

≡ Network Constants

Access Network framework constants used in C.

≡ Network Functions

Access Network framework functions used in C.

≡ Network Data Types

Protocols

```
protocol BrowserProvider
```

BrowserProviders can be used when creating NetworkBrowsers.

```
protocol Connectable
```

Describes types that can be used to make NetworkConnections.

`protocol ConnectionStorage`

Types that conform to ConnectionStorage can be used as additional storage within a connection.

`protocol DatagramProtocol`

Types that conform to DatagramProtocol send and receive messages with minimal or no metadata, usually constrained to a fixed maximum size.

`protocol FramerProtocol`

Framer protocols allow custom framing and serialization of messages on a connection.

`protocol ListenerProvider`

Extensible support for configuring advertise descriptors to define the service a listener should advertise.

`protocol MessageProtocol`

Types that conform to MessageProtocol send and receive messages. The conforming type is responsible for specifying its message-specific metadata.

`protocol MultiplexProtocol`

Types that conform to MultiplexProtocol are allowed to be the top protocol in a network protocol stack for multiplexing network connection objects.

`protocol NWParametersProvider`

Types that conform to the NWParametersProvider protocol can be used to generate an NWParameters.

`protocol NetworkCoder`

`protocol NetworkDecoder`

A type that conforms to the NetworkEncoder protocol can decode data to an Encodable object

`protocol NetworkEncoder`

A type that conforms to the NetworkEncoder protocol can encode a Encodable object to Data

`protocol NetworkFixedWidthInteger`

`protocol NetworkMetadataProtocol`

Types that conform to NetworkProtocolOptions can be used when configuring protocol stacks.

`protocol NetworkProtocolOptions`

`protocol OneToOneProtocol`

Types that conform to `OneToOneProtocol` are allowed to be the top protocol in a network protocol stack for non-multiplexed connections.

`protocol StreamProtocol`

Types that conform to the `StreamProtocol` protocol expose methods for sending and receiving byte streams.

Variables

`let kNSErrorDomainWiFiAware: CFString`

`var nw_error_domain_wifi_aware: nw_error_domain_t`

`var nw_link_quality_good: nw_link_quality_t`

`var nw_link_quality_minimal: nw_link_quality_t`

`var nw_link_quality_moderate: nw_link_quality_t`

`var nw_link_quality_unknown: nw_link_quality_t`

Functions

`func nw_parameters_get_allow_ultra_constrained(nw_parameters_t) -> Bool`

`func nw_parameters_set_allow_ultra_constrained(nw_parameters_t, Bool)`

`func nw_path_get_link_quality(nw_path_t) -> nw_link_quality_t`

`func nw_path_is_ultra_constrained(nw_path_t) -> Bool`

`func withNetworkConnection<ApplicationProtocol>(to: NWEndpoint, using:
() -> ApplicationProtocol, (NetworkConnection<ApplicationProtocol>)
async throws -> Void) async throws`

`func withNetworkConnection<ApplicationProtocol>(to: NWEndpoint, using:
() -> ApplicationProtocol, (NetworkConnection<ApplicationProtocol>)
async throws -> Void) async throws`

`func withNetworkConnection<ApplicationProtocol>(to: NWEndpoint, using:
NWParametersBuilder<ApplicationProtocol>, (NetworkConnection<
ApplicationProtocol>) async throws -> Void) async throws`

```
func withNetworkConnection<ApplicationProtocol>(to: NWEndpoint, using:  
NWParametersBuilder<ApplicationProtocol>, (NetworkConnection<  
ApplicationProtocol>) async throws -> Void) async throws
```