Hazel Networking Protocol

**Specification**

Version 1.0

Jamie Read

# Table of Contents

Contents

[Table of Contents 2](#_Toc470974992)

[Introduction 3](#_Toc470974993)

[Shared Attributes and General Notes 4](#_Toc470974994)

[Terminology 4](#_Toc470974995)

[Interfaces 4](#_Toc470974996)

[Handshaking 4](#_Toc470974997)

[Version 0 4](#_Toc470974998)

[Implementation Notes 4](#_Toc470974999)

[TCP Implementation 5](#_Toc470975000)

[Connecting 5](#_Toc470975001)

[Packet Splitting 5](#_Toc470975002)

[Disconnection 5](#_Toc470975003)

# Introduction

Hazel Networking is a low-level networking library for C# providing connection orientated, message based communication via TCP, UDP and RUDP.

Its aim is to provide a standardized interface for web communication so that using and switching between protocols is incredibly simple.

Hazel is going to be the basis of DarkRift 2 and it is being released completely open source so that members of the community can make use of it, improve it and help find any bugs before DarkRift 2 is released.

If you feel that anything in this document is not explained fully or you would like to make other comments I can be reached at jamie@darkriftnetworking.com.

# Shared Attributes and General Notes

Big endian (Network) order

## Terminology

* **Application** – The library, program, application etc. that is using Hazel to send/receive messages.

## Interfaces

And Events

## Handshaking

Hazel implements a standardized handshake across all protocols, this allows the client application to send data that is received by the server as part of the new connection event rather than as a standard message received event.

The first byte of the handshake is a version number specifying what subsequent bytes detail.

### Version 0

|  |  |
| --- | --- |
| 1 byte |  |
| Version | Data |

Figure 1 The layout of a standard Hazel handshake.

Where the length of the Data field expands to fill the message, e.g. for a message of length 100 bytes there would be 99 bytes of data.

## Implementation Notes

It is not necessary to implement all transport methods outlined in this document, it is sufficient to only implement those needed for the specific application. In fact, it would probably be beneficial to implement each within its own library allowing other users to only reference and include those protocols they require.

# TCP Implementation

The TCP implementation is likely to be the simplest transport method to implement in Hazel. As the TCP protocol already provides guarantees of reliability and fragmentation very little work is required to convert for Hazel compatibility.

## Connecting

Connections with TCP can simply use the default TCP implementation of the operating system to form connections. It is important that a standard Hazel handshake message is written to the TCP input stream immediately after connection has been established and before any other messages are sent.

The handshake should be made up of a 4-byte length header as outlined in the [Packet Splitting](#_Packet_Splitting) section and a standard Hazel handshake, including version byte.

|  |  |
| --- | --- |
| 4 bytes |  |
| Length | Standard Hazel handshake |

Figure 2 The layout of a TCP handshake within Hazel.

Where the Standard Hazel Handshake field is of length Length.

## Packet Splitting

As TCP is stream based there must be a distinction between packets. For this Hazel uses 4 bytes before each packet to denote the length of the packet (excluding the 4-byte length itself). This is always included, even for handshakes.

|  |  |
| --- | --- |
| 4 bytes |  |
| Length | Data |

Figure 3 Table showing how the length header precedes the data of the message.

Where the Data field is of length Length.

As an example, if a message contained 100 bytes of data the 4 Length bytes would be hold the value 100 and the total length of the packet would be 104 bytes.

## Disconnection

In the TCP implementation, it is sufficient to use TCP’s built in disconnection using the shutdown flag.