Zenet Docs

Zenet Library

Introduction

What is zenet?

Zenet is a socket library. This is abstract and facilitates real-time (online) application creation and development using socket. Its strong point or motivation of creation is the ease of use and speed, that is without external dependencies and without much overhead and it is extensible.

How to use or use “Zenet”?

In this document is the description and documentation of its use.

Runtime and supported platforms

Zenet runs and supports all platforms, which runs dotnet the solution is based on the “dotnet standard 2.0 - library“ version of dotnet. Whether in “dll” and/or “bare source code”

Unity runtime

As described the current library runs on all “dotnet” platforms and unity does not shy away from it! We even have a library available in the asset store

Warning

Depending on the platform, the runtime needs to dispatch the messages on its own, relax this is so simple that it only takes 2 lines of code “1 line: it will say that the responsibility of dispatching the messages is manual. This means that when receiving a message it is not dispatched automatically but manually” and “2nd line: it must be executed in an update or loop, so that each message received is dispatched in the dispatcher loop”, that’s all. necessary for runtimes like Unity, FlaxEngine, etc. but is already built in (Unity Asset Store).

See the tutorial in code, in the topic (“Main Thread“).

Documentation

Topics

- Instance creation
- Open connection
- Close connection
- Event notification
- Send data

- Send event
- ZCallback (Main Thread / Callback)
- ZAsync (Async)
- Socket customization

Instance creation

- **Unity**

  –

  ```
  using Zenet.Tcp;

  public class Example : MonoBehaviour
  {
      public TcpClient client;

      private void Awake()
      {
          // create a instance
          client = new TcpClient();
      }
  }
  ```

  –

  ```
  using Zenet.Tcp;

  public class Example : MonoBehaviour
  {
      public TcpServer server;

      private void Awake()
      {
          // create a instance
          server = new TcpServer();
      }
  }
  ```

-

Dotnet 6

–

```
using Zenet.Tcp;

// create a instance
TcpClient client = new();
```

–

```
using Zenet.Tcp;

// create a instance
TcpServer server = new();
```

Open connection

- **Unity**

  –

```
using Zenet.Tcp;
using Zenet.Unity;

// automatically create "ZenetHost" when connecting this script to object
[RequireComponent(typeof(ZenetHost))]
public class Example : MonoBehaviour
{
    public ZenetHost host;
    public TcpClient client;

    private void Awake()
    {
        // get instance of ZenetHost
        host = GetComponent<ZenetHost>();
```

```
        // create a instance
        client = new TcpClient();

        // open connection
        client.Open(host.GetHost());
    }
}
```

–

```
using Zenet.Tcp;

public class Example : MonoBehaviour
{
    public ZenetHost host;
    public TcpServer server;

    private void Awake()
    {
        // get instance of ZenetHost
        host = GetComponent<ZenetHost>();

        // create a instance
        server = new TcpServer();

        // open connection
        server.Open(host.GetHost());
    }
}
```

- **Dotnet 6**

  –

```
using Zenet.Tcp;

// create a instance
TcpClient client = new();

// create endpoint host
```

```
ZHost host = new("127.0.0.1", 8080);

// open connection
client.Open(host);
```

–

```
using Zenet.Tcp;

// create a instance
TcpServer server = new();

// create endpoint host
ZHost host = new("127.0.0.1", 8080);

// open connection
server.Open(host);
```

Close connection

- *** Plataform**

  –

```
// Warning: assuming you already have the TcpClient creating and calling client

// close connection
client.Close();
```

–

```
// Warning: assuming you already have the TcpServer creating and calling server

// close connection
server.Close();
```

Event notification

- *** Plataform**

  –

```
// Warning: assuming you already have the TcpClient creating and calling client

// is called when it receives the connection opening event
client.OnOpen(OnOpenHandle);

/* receives the connection open error event.
    parameters:
        (Exception e) - connection opening attempt exception */
client.OnError(OnErrorHandle);

// is called when it receives the connection closing event
client.OnClose(OnCloseHandle);

/* is called when it receives a data/message.
    parameters:
        ( byte[] data ) - the received data */
client.OnData(OnDataHandle);

/* is called when it receives a "Zenet Event".
    parameters:
    (string name) - the name of the received event "is marked when the event is sent"
    ( byte[] )    - the received data */
client.OnEvent(OnEventHandle);

void OnOpenHandle()
{
    Console.WriteLine("connection opened!");
}

void OnErrorHandle(Exception e)
{
    Console.WriteLine($"error on open connection: {e}");
}

void OnCloseHandle()
```

```
{
    Console.WriteLine($"connection closed");
}

void OnDataHandle(byte[] data)
{
    Console.WriteLine($"data received: {ZEncoding.ToString(data)}");

    // echo a data
    client.ToData(data);
}

void OnEventHandle(string name, byte[] data)
{
    Console.WriteLine($"event received ({name}): {ZEncoding.ToString(data)}");

    // echo a event
    client.ToEvent(name, data);
}
```

—

```
// Warning: assuming you already have the TcpServer creating and calling server

// is called when it receives the connection opening event
server.OnOpen(OnOpenHandle);

/* receives the connection open error event.
    parameters:
        (Exception e) - connection opening attempt exception */
server.OnError(OnErrorHandle);

// is called when it receives the connection closing event
server.OnClose(OnCloseHandle);

/* is called when a client enter "connect" to the server.
    parameters:
        ( TcpClient client ) - the client instance on the server "the client's mirror c
server.OnEnter(OnEnterHandle);

/* is called when a client exit "disconnect" to the server.
    parameters:
        ( TcpClient client ) - the client instance on the server "the client's mirror c
```

```
server.OnExit(OnExitHandle);

/* is called when it receives a data/message.
    parameters:
        ( TcpClient client) - the client instance on the server "the client's mirror or
        ( byte[] data ) - the received data */
server.OnData(OnDataHandle);

/* is called when it receives a "Zenet Event".
    parameters:
    ( TcpClient client) - the client instance on the server "the client's mirror on the
    (string name) - the name of the received event "is marked when the event is sent"
    ( byte[] )    - the received data */
server.OnEvent(OnEventHandle);

void OnOpenHandle()
{
    Console.WriteLine("connection opened!");
}

void OnErrorHandle(Exception e)
{
    Console.WriteLine($"error on open connection: {e}");
}

void OnCloseHandle()
{
    Console.WriteLine($"connection closed");
}

void OnEnter(TcpClient client)
{
    Console.WriteLine($"client enter, id {client.Id}");
}

void OnExit(TcpClient client)
{
    Console.WriteLine($"client exit, id {client.Id}");
}

void OnDataHandle(TcpClient client, byte[] data)
{
    Console.WriteLine($"data received: {ZEncoding.ToString(data)}");

    // echo a data
    client.ToEvent(data);
```

```
}

void OnEventHandle(TcpClient client, string name, byte[] data)
{
    Console.WriteLine($"event received ({name}): {ZEncoding.ToString(data)}");

    // echo a event
    client.ToEvent(name, data);
}
```

Send data

- *** Plataform**

  –

```
// Warning: assuming you already have the TcpClient creating and calling client

// convert a string to bytes (byte[])
byte[] messageOrData = ZEncoding.ToBytes("hello server!");

// send a data to the server
client.ToData(messageOrData);
```

  –

```
// Warning: assuming you already have the TcpServer creating and calling server

server.OnData(OnDataHandle);

void OnDataHandle(TcpClient client, byte[] data)
{
    // convert a string to bytes (byte[])
    byte[] messageOrData = ZEncoding.ToBytes("hello client! [GENERIC]");

    // send a data to the client
    client.ToData(messageOrData);
}
```

you can also use lambda expressions “‘csharp // Warning: assuming you already have the TcpServer creating and calling server

server.OnData((TcpClient client, byte[] data) => { // convert a string to bytes (byte[]) byte[] messageOrData = ZEncoding.ToBytes(“hello client! [LAMBDA]”);

```
  // send a data to the client
  client.ToData(messageOrData);
```

}); “‘

Send event

- *** Plataform**

  –

```
// Warning: assuming you already have the TcpClient creating and calling client

// convert a string to bytes (byte[])
byte[] messageOrData = ZEncoding.ToBytes("hello server!");

// send a event to the server
client.ToEvent("welcome", messageOrData);
```

  –

```
// Warning: assuming you already have the TcpServer creating and calling server

server.OnData(OnEventHandle);

void OnEventHandle(TcpClient client, string name, byte[] data)
{
    if (name == "welcome")
    {
        // convert a string to bytes (byte[])
        byte[] messageOrData = ZEncoding.ToBytes("hello client! [GENERIC]");
```

```
        // send a event to the client
        client.ToEvent("welcome", messageOrData);
    }
    else
    {
        // convert a string to bytes (byte[])
        byte[] messageOrData = ZEncoding.ToBytes("there's no way to answer you! [GENERI

        // send a error to client event
        client.ToEvent("welcome error", messageOrData);
    }
}
```

you can also use lambda expressions “‘csharp // Warning: assuming you already have the TcpServer creating and calling server

server.OnData((TcpClient client, string name, byte[] data) => { if (name == “welcome”) { // convert a string to bytes (byte[]) byte[] messageOrData = ZEncoding.ToBytes(“hello client! [LAMBDA]”);

```
      // send a event to the client
      client.ToEvent("welcome", messageOrData);
  }
  else
  {
      // convert a string to bytes (byte[])
      byte[] messageOrData = ZEncoding.ToBytes("there's no way to answer you! [LAMBDA]"

      // send a error to client event
      client.ToEvent("welcome error", messageOrData);
  }
```

}); “‘

ZCallback (Main Thread / Callback)

- *** Plataform #### This example is based on unity, but it works on all platforms the same way.** #### This is already built into the asset store version of the unit in an improved way and is created automatically as soon as you create “ZenetHost”! The Zenet “API” use callback functions to return their execution results. The various callback functions are called from threads other than the main thread. However,

Unity requires function calls or actions (such as UI manipulation) to come from the main thread.

To solve this problem, use the Unity Main Thread script with your Game content to dispatch callback functions to the main thread. Here's how:

```
using Zenet.Core;
using UnityEngine;

public class MainThread : MonoBehaviour
{
    private void Awake()
    {
        // set manual dispatch
        ZCallback.Manual = true;
    }

    private void Update()
    {
        // dispatch the events
        ZCallback.Update();
    }
}
```

You can also use and enjoy this implementation see the example below
“‘csharp using Zenet.Core; using UnityEngine; using UnityEngine.UI;

public class Example : MonoBehaviour { private void Start() { // ……. code …….

```
        // "create ThreadPool" (thread) to perform "heavy" tasks in the background
        ZAsync.Execute(() =>
        {
            // random number
            int number = Random.Range(0, 100000000);

            // (index) is 100 million
            int index = 100000000;

            while(index > 0)
            {
                // check if "index" equals "number"
                if (index == number)
                {
                    // execute action on main thread
                    ZCallback.Execute(() =>
                    {
```

```
                        Button button = gameObject.GetComponet<Button>();
                        button.interactable = false;
                    });

                    break;
                }

                // decreasing the (index)
                index--;
            }
        });
    }

}
```

Async (ZAsync)

- *** Plataform** #### **“Async or Z Async” is an abstraction that allows you to run your code in the background** ie your code is running at the same time as your other code! and you are many if you will need to use “MainThread” and “ZCallback.Execute()” to be able to execute your code in the main Thread it is very interesting to use this to perform heavy tasks, … “‘csharp using Zenet.Core;

  ZAsync.Execute(() => { // …… here is async ……

  ```
  int i = 0;

  // max is 10 milion
  int max = 10000000;

  while( i < max)
  {
      i++;
  }

  ZCallback.Execute(() =>
  {
      Debug.Log($"current (i) is {i}");
  });
  ```

  }); “‘

Socket customization

-

*** Plataform**

–

```
using System;
using System.Net;
using System.Net.Sockets;

// Warning: assuming you already have the TcpClient creating and calling client

client.BeforeOpen((Socket socket) =>
{
    // more: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.send

    // Don't allow another socket to bind to this port.
    socket.ExclusiveAddressUse = true;

    // Disable the Nagle Algorithm for this tcp socket.
    socket.NoDelay = true;
});

client.AfterOpen((Socket socket) =>
{
    // more: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.send

    // Set the timeout for synchronous receive methods to
    // 1 second (1000 milliseconds.)
    socket.ReceiveTimeout = 1000;

    // Set the timeout for synchronous send methods
    // to 1 second (1000 milliseconds.)
    socket.SendTimeout = 1000;
});
```

–

```
using System;
using System.Net;
using System.Net.Sockets;

// Warning: assuming you already have the TcpServer creating and calling server
```

```
server.BeforeOpen((Socket socket) =>
{
    // more: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.send

    // Don't allow another socket to bind to this port.
    socket.ExclusiveAddressUse = true;

    // Disable the Nagle Algorithm for this tcp socket.
    socket.NoDelay = true;
});

server.AfterOpen((Socket socket) =>
{
    // more: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.socket.send

    // Set the timeout for synchronous receive methods to
    // 1 second (1000 milliseconds.)
    socket.ReceiveTimeout = 1000;

    // Set the timeout for synchronous send methods
    // to 1 second (1000 milliseconds.)
    socket.SendTimeout = 1000;
});
```