Exercises in .NET with Andras Nemes

Tips and tricks in C# .NET

Basics of working with pipes in C# .NET part 4: basic conversation with messages

JUNE 23, 2015 2 COMMENTS (HTTPS://DOTNETCODR.COM/2015/06/23/BASICS-OF-WORKING-WITH-PIPES-IN-C-NET-PART-4-BASIC-CONVERSATION-WITH-MESSAGES/#COMMENTS)

In this (https://dotnetcodr.com/2015/06/16/basics-of-working-with-pipes-in-c-net-part-1-send-and-receive-a-single-byte/) post we saw how a pipe stream client and server can send each other single bytes. In this (https://dotnetcodr.com/2015/06/19/basics-of-working-with-pipes-in-c-net-part-3-message-transmission/) post we managed to send a single message from the client to the server. It's time to marry the two concepts. We'll let the client and server start a conversation.

The server side code has nothing new compared to the posts referred to above:

```
private static void ConversationWithTheClient()
{
    using (NamedPipeServerStream namedPipeServer = new NamedPipeServerStream("test-pipe", PipeDirecti
        1, PipeTransmissionMode.Message))
    {
        Console.WriteLine("Server waiting for a connection...");
        namedPipeServer.WaitForConnection();
        Console.Write("A client has connected, send a greeting from the server: ");
        string message = Console.ReadLine();
        byte[] messageBytes = Encoding.UTF8.GetBytes(message);
        namedPipeServer.Write(messageBytes, 0, messageBytes.Length);

        string response = ProcessSingleReceivedMessage(namedPipeServer);
        Console.WriteLine("The client has responded: {0}", response);
        while (response != "x")
        {
            Console.Write("Send a response from the server: ");
            message = Console.ReadLine();
            messageBytes = Encoding.UTF8.GetBytes(message);
            namedPipeServer.Write(messageBytes, 0, messageBytes.Length);
            response = ProcessSingleReceivedMessage(namedPipeServer);
            Console.WriteLine("The client is saying {0}", response);
        }

        Console.WriteLine("The client has ended the conversation.");
    }
}

private static string ProcessSingleReceivedMessage(NamedPipeServerStream namedPipeServer)
{
    StringBuilder messageBuilder = new StringBuilder();
    string messageChunk = string.Empty;
    byte[] messageBuffer = new byte[5];
    do
    {
        namedPipeServer.Read(messageBuffer, 0, messageBuffer.Length);
        messageChunk = Encoding.UTF8.GetString(messageBuffer);
        messageBuilder.Append(messageChunk);
        messageBuffer = new byte[messageBuffer.Length];
    }
    while (!namedPipeServer.IsMessageComplete);
    return messageBuilder.ToString();
}
```

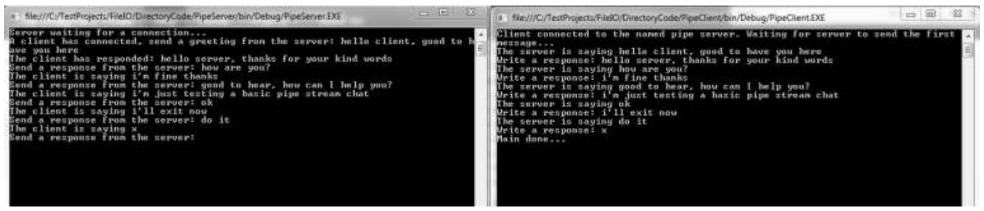
Call ConversationWithTheClient from Main in the server console app. The client side code has a couple of new features:

```
private static void ConversationWithTheServer()
{
    using (NamedPipeClientStream namedPipeClient = new NamedPipeClientStream(".", "test-pipe", PipeDi
    {
        namedPipeClient.Connect();
        Console.WriteLine("Client connected to the named pipe server. Waiting for server to send the
        namedPipeClient.ReadMode = PipeTransmissionMode.Message;
        string messageFromServer = ProcessSingleReceivedMessage(namedPipeClient);
        Console.WriteLine("The server is saying {0}", messageFromServer);
        Console.Write("Write a response: ");
        string response = Console.ReadLine();
        byte[] responseBytes = Encoding.UTF8.GetBytes(response);
        namedPipeClient.Write(responseBytes, 0, responseBytes.Length);
        while (response != "x")
        {
            messageFromServer = ProcessSingleReceivedMessage(namedPipeClient);
            Console.WriteLine("The server is saying {0}", messageFromServer);
            Console.Write("Write a response: ");
            response = Console.ReadLine();
            responseBytes = Encoding.UTF8.GetBytes(response);
            namedPipeClient.Write(responseBytes, 0, responseBytes.Length);
        }
    }
}

private static string ProcessSingleReceivedMessage(NamedPipeClientStream namedPipeClient)
{
    StringBuilder messageBuilder = new StringBuilder();
    string messageChunk = string.Empty;
    byte[] messageBuffer = new byte[5];
    do
    {
        namedPipeClient.Read(messageBuffer, 0, messageBuffer.Length);
        messageChunk = Encoding.UTF8.GetString(messageBuffer);
        messageBuilder.Append(messageChunk);
        messageBuffer = new byte[messageBuffer.Length];
    }
    while (!namedPipeClient.IsMessageComplete);
    return messageBuilder.ToString();
}
```

The "." in the constructor indicates the local machine. There's no NamedPipeClientStream constructor where you can only set the pipe name and the pipe direction. The pipe transmission mode can be set after the NamedPipeClientStream has been constructed using the ReadMode property.

Call ConversationWithTheServer from Main of the client console application. Run both apps and you can start a simple conversation like this:


file:///C:/TestProjects/FileIO/DirectoryCode/PipeServer/bin/Debug/PipeServer.EXE
file:///C:/TestProjects/FileIO/DirectoryCode/PipeClient/bin/Debug/PipeClient.EXE

```
Server waiting for a connection...
A client has connected, send a greeting from the server: hello client, good to h
ave you here
The client has responded: hello server, thanks for your kind words
Send a response from the server: how are you?
The client is saying i'm fine thanks
Send a response from the server: good to hear, how can I help you?
The client is saying i'm just testing a basic pipe stream chat
Send a response from the server: ok
The client is saying i'll exit now
Send a response from the server: do it
The client is saying x
Send a response from the server:
```

```
Client connected to the named pipe server. Waiting for server to send the first
message...
The server is saying hello client, good to have you here
Write a response: hello server, thanks for your kind words
The server is saying how are you?
Write a response: i'm fine thanks
The server is saying good to hear, how can I help you?
Write a response: i'm just testing a basic pipe stream chat
The server is saying ok
Write a response: i'll exit now
The server is saying do it
Write a response: x
Main done...
```

(https://dotnetcodr.files.wordpress.com/2015/03/pipe-stream-simple-conversation.png)

Read the next part here (https://dotnetcodr.com/2015/06/24/basics-of-working-with-pipes-in-c-net-part-5-transmitting-objects/).

View the list of posts on Messaging here (https://dotnetcodr.com/messaging/).
FILED UNDER .NET, MESSAGING TAGGED WITH C#, MESSAGING, PIPE

About Andras Nemes
I'm a .NET/Java developer living and working in Stockholm, Sweden.

2 Responses to *Basics of working with pipes in C# .NET part 4: basic conversation with messages*

Michal says:
June 2, 2016 at 12:00 pm
thanks a lot, finally easy to understand explanation of how named pipes works

Reply

Matt says:
January 19, 2018 at 8:22 pm
I think there may be a bug: Since UTF8 can have multiple bytes in a character, it seems the byte buffer might end with an incomplete character when you try to decode it (and the next read would begin with a continuation character which can't be decoded).

Reply