Basics of working with pipes in C# .NET part 5: transmitting objects

JUNE 24, 2015 1 COMMENT (HTTPS://DOTNETCODR.COM/2015/06/24/BASICS-OF-WORKING-WITH-PIPES-IN-C-NET-PART-5-TRANSMITTING-OBJECTS/#COMMENTS)

In the previous (https://dotnetcodr.com/2015/06/23/basics-of-working-with-pipes-in-c-net-part-4-basic-conversation-with-messages/) part we saw a very basic chat application between a client and a server through a pipe. In this post we'll see a way of how to transmit objects from the client to the server.

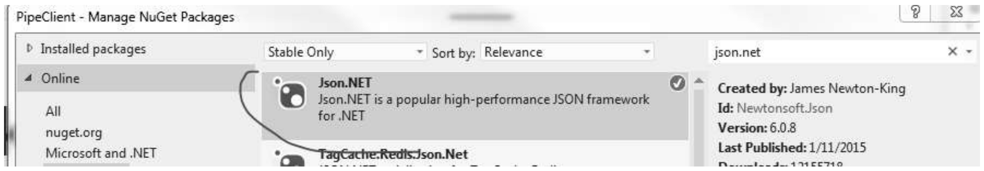
You'll see that it really is not much different from transmitting a message. We cannot simply transmit an object using e.g. a WriteObject method. Instead the object must be serialised at the client side and deserialised at the server side. We'll serialise our object using JSON and transmit the object properties using the message transmission technique we've seen earlier. The server will then deserialise the JSON string into the object.

We'll pretend that the client wants to transmit an order:

```
public class Order
{
    public string ProductName { get; set; }
    public int Quantity { get; set; }
    public string CustomerName { get; set; }
    public string Address { get; set; }
}
```

The client and server applications can both have access to this object. You can save it in both console applications to simulate that the server and client apps are completely decoupled. Alternatively you can create a shared C# class library that both server and client can refer to.

We'll also need a JSON library and an obvious choice is the popular JSON.NET dll which can be downloaded from NuGet:



Json.NET is a popular high-performance JSON framework for .NET

Created by: James Newton-King
Id: Newtonsoft.Json
Version: 6.0.8
Last Published: 1/11/2015

(https://dotnetcodr.files.wordpress.com/2015/03/json-net-library-in-nuget.png)

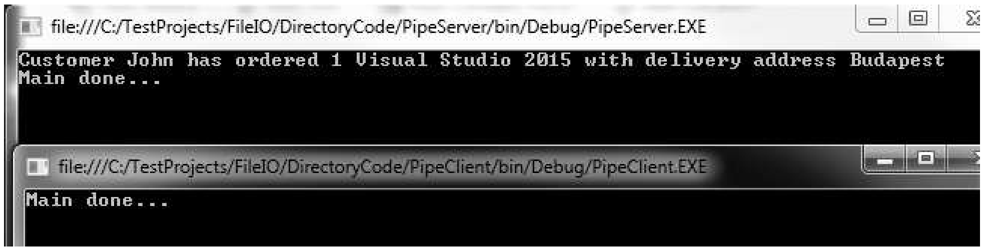
The client side code is almost identical to what we saw before in this series. The only new thing is the object serialisation:

```
private static void SendOrderToServer()
{
    using (NamedPipeClientStream namedPipeClient = new NamedPipeClientStream("orders"))
    {
        Order order = new Order()
        {
            Address = "Budapest",
            CustomerName = "John",
            ProductName = "Visual Studio 2015",
            Quantity = 1
        };
        string serialised = JsonConvert.SerializeObject(order);
        namedPipeClient.Connect();
        byte[] messageBytes = Encoding.UTF8.GetBytes(serialised);
        namedPipeClient.Write(messageBytes, 0, messageBytes.Length);
    }
}
```

The server side code also only has one new feature, which is the object deseralisation:

```
private static void ReceiveObjectFromClient()
{
    using (NamedPipeServerStream namedPipeServer = new NamedPipeServerStream("orders", PipeDirection.
        1, PipeTransmissionMode.Message))
    {
        namedPipeServer.WaitForConnection();
        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);
        Order order = JsonConvert.DeserializeObject<Order>(messageBuilder.ToString());
        Console.WriteLine("Customer {0} has ordered {1} {2} with delivery address {3}", order.Custome
    }
}
```

As usual call these methods from Main of the server and client console applications and run them both in Visual Studio. You should see that the server receives the order correctly:


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

```
Customer John has ordered 1 Visual Studio 2015 with delivery address Budapest
Main done...
```

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

```
Main done...
```

(https://dotnetcodr.files.wordpress.com/2015/03/order-object-transmitted-from-client-to-server-through-pipe.png)

Read the next and last part of this series here (https://dotnetcodr.com/2015/06/26/basics-of-working-with-pipes-in-c-net-part-6-message-compression/) which discusses message compression.

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.

One Response to *Basics of working with pipes in C# .NET part 5: transmitting objects*

Paul.Matthews says:
April 12, 2017 at 2:13 pm
Thanks for the blog – this particular feature I wanted to use and needed someone to show me, so thanks again.
Kind regards
Paul

Reply