Basics of working with pipes in C# .NET part 6: message compression

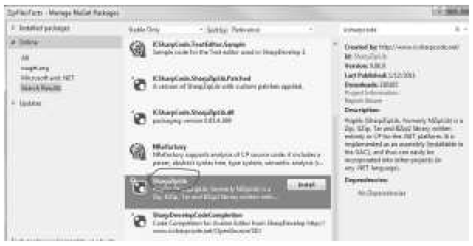
JUNE 26, 2015 4 COMMENTS (HTTPS://DOTNETCODR.COM/2015/06/26/BASICS-OF-WORKING-WITH-PIPES-IN-C-NET-PART-6-MESSAGE-COMPRESSION/#COMMENTS)

In the previous (https://dotnetcodr.com/2015/06/24/basics-of-working-with-pipes-in-c-net-part-5-transmitting-objects/) part we looked at a possible solution to transmit objects through a pipe from the client to a server. We saw that it was not much different compared to transmitting a string message as the object had to be serialised first.

In this post we'll consider a way to reduce the message size by compressing it. In messaging you should always aim for short and concise messages for good speed and scaling. However, sometimes the messages are simply large and you cannot do much about it. E.g. consider a service that scans the contents of a web page and returns the HTML as a string along with some measurements for every element on the page, such as CSS and JS files etc: time to first byte, header receive time, header size etc. That can be a large piece of string.

We'll reuse a technique outlined in this (https://dotnetcodr.com/2015/01/30/compressing-and-decompressing-strings-with-bzip2-in-net-c/) post dedicated to BZip2 string compression. If you don't know how that works then read through that article, it's very short.

You'll need to import the following NuGet package to use BZip2 in the pipe server in client console applications:


(https://dotnetcodr.files.wordpress.com/2014/10/sharpziplib-nuget.png)

Here's the pipe client code to send a BZip2 compressed message to the pipe server:

```
private static void SendCompressedMessageToServer()
{
    using (NamedPipeClientStream namedPipeClient = new NamedPipeClientStream("test"))
    {
        string longMessage = "This is a large piece of text";
        string compressed = string.Empty;
        using (MemoryStream source = new MemoryStream(Encoding.UTF8.GetBytes(longMessage)))
        {
            using (MemoryStream target = new MemoryStream())
            {
                BZip2.Compress(source, target, true, 4096);
                byte[] targetByteArray = target.ToArray();
                compressed = Convert.ToBase64String(targetByteArray);
            }
        }
        namedPipeClient.Connect();
        byte[] messageBytes = Encoding.UTF8.GetBytes(compressed);
        namedPipeClient.Write(messageBytes, 0, messageBytes.Length);
    }
}
```

…and here's how the server can handle the message:

```
private static void ReceiveCompressedMessageFromClient()
{
    using (NamedPipeServerStream namedPipeServer = new NamedPipeServerStream("test", PipeDirection.In
        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);
        string fullMessage = messageBuilder.ToString().Trim('\0');
        byte[] largeCompressedTextAsBytes = Convert.FromBase64String(fullMessage);
        using (MemoryStream source = new MemoryStream(largeCompressedTextAsBytes))
        {
            using (MemoryStream target = new MemoryStream())
            {
                BZip2.Decompress(source, target, true);
                string uncompressedString = Encoding.UTF8.GetString(target.ToArray());
                Console.WriteLine(uncompressedString);
            }
        }
    }
}
```

We compress the string using the BZip2 algorithm and then convert that to a base 64 string in the client. The server performs the exact opposite. You may be wondering about this bit:

```
string fullMessage = messageBuilder.ToString().Trim('\0');
```

The character described in the Trim argument is the string termination or null-termination character that marks the end of the message. That's how the pipe server and client can determine whether there's anything remaining of an incoming message.

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.

4 Responses to *Basics of working with pipes in C# .NET part 6: message compression*

SutoCom says:
June 27, 2015 at 2:35 pm
Reblogged this on SutoCom Solutions.

Reply
Marcelo Matt says:
June 6, 2016 at 3:14 pm
Andras, how are you?

Well, I'm fine.

I have a question: Is it possible to use pipes with webAPI? If the answer is Yes, could you make a post with an example?

BTW, tks for the excellent work.

See ya.

Reply

bagwani@hotmail.com says:
January 26, 2017 at 2:28 pm
Nice work Andras,

Do you know LZ4 compression? It has very good compression and thoughput characteristic

Check the follwoing C3 implementation
https://github.com/MiloszKrajewski/lz4net

Cheers,

Reply

Andras Nemes says:
January 26, 2017 at 9:18 pm
Hello, no, never hear of, thanks for the tip. //Andras

Reply