Cloudy Message Passing Library

Documentation

The Cloudy Message Passing Library is a .NET library for development of scalable parallel applications.

Pavel Perestoronin

10/21/2011

1 Overview

The library consists of the following separate parts interacting with one another:

- Protobuf namespace (Protocol Buffers implementation)
- Messaging namespace
- Connections namespace
- Helper classes

2 Components

2.1 Protocol Buffers – the Protobuf namespace

2.1.1 Getting Started

In order to serialize an object of the specific class you should firstly mark this class with the `ProtobufSerializable` attribute and each serializable field – with the `ProtobufField` attribute:

```
[ProtobufSerializable]
public class A
{
    /// <summary>
    /// Initializes the default values.
    /// </summary>
    public A()
    {
        B = 666;
    }

    [ProtobufField(1)]
    public uint B { get; set; }
}
```

Then you'll be able to serialize an object by creating the serializer and calling the Serialize method and deserialize calling the Deserialize method:

```
[Test]
public void TestSerializeBasic()
{
    Serializer serializer = Serializer.CreateSerializer(typeof(A));
    object o = new A { B = 150 };
    AssertExtensions.AreEqual(new byte[] { 0x08, 0x96, 0x01 },
            serializer.Serialize(o));
}
```

2.1.2 Optional and Required Fields

All properties are optional by default. This means that if a field has no value set then the related tag will not appear in a target message. This behavior is recommended because you'll not be able to remove a required field and not break a protocol.

But the possibility to define a required field there is:

```
[ProtobufField(1, required: true)]
public string D { get; set; }
```

2.1.3 Repeated Fields

The Cloudy can serialize collections. All you need is to define a property as `ICollection`:

```
[ProtobufField(1)]
public ICollection<uint> List { get; set; }
...
Serializer serializer = Serializer.CreateSerializer(typeof(D));
object o = new D { List = new uint[] { 1, 2, 3 } };
AssertExtensions.AreEqual(new byte[] { 0x08, 0x01, 0x08, 0x02, 0x08, 0x03 },
    serializer.Serialize(o));
```

2.1.4 Packed Repeated Fields

Packed repeated field is serialized as length-delimited field: sequentially serialized values are used instead of repeating of a single tag with a single value.

```
[ProtobufSerializable]
public class E
{
    [ProtobufField(4, packed: true)]
    public ICollection<uint> List { get; set; }
}
```

2.1.5 Types Mapping

By default the .NET types are serialized into the following Protobuf types:

.NET Type	Protobuf Type
bool	Unsigned Varint
int	Signed Varint
long	Signed Varint
uint	Unsigned Varint
ulong	Unsigned Varint
string	String
byte[]	Length-Delimited
Guid	Length-Delimited (16 bytes)
Enum	Unsigned Varint
ICollection<T>	Repeated T
Nullable<T>	Optional T
Any other class	Attempted to be serialized as an Embedded Message

If you want to change a target Protobuf type (e.g. serialize int as Fixed32) then you may specify the `dataType` parameter of the `ProtobufSerializable` attribute:

```
[ProtobufSerializable]
public class H
{
    [ProtobufField(2, dataType: DataType.FixedInt32)]
    public int Fixed32 { get; set; }
}
```

Data types are mapped into the target Protobuf types as follows:

DataType	Protobuf Type
Bool	Varint
Bytes	Length-Delimited
Embedded Message	Length-Delimited
FixedInt32	Fixed32
FixedInt64	Fixed64
FixedUInt32	Fixed32
FixedUInt64	Fixed64

SignedVarint	Signed Varint
String	String
UnsignedVarint	Varint
Guid	Length-Delimited (16 bytes)

2.2 Messaging Utility Classes – the Messaging namespace

2.2.1 MessageStream

This is the utility class for convenient sequential reading and writing of messages. Wraps a Stream object and provides the Read and Write methods. Thread-safe.

2.2.1.1 Example

```
using (MemoryStream stream = new MemoryStream())
{
    MessageStream messageStream = new MessageStream(stream);
    foreach (object message in
        new object[] { new A { B = 1 }, new A { B = 2 } })
    {
        messageStream.Write(message);
    }
}
```

2.3 Connectivity – the Connections namespace

2.3.1 Understanding DTO's

2.4 Helper Classes – the Helpers namespace

2.4.1 UdpStream

Implements the Stream interface. That allows interacting with an UDP connection as if it was simply a Stream. This is useful in UDP-messaging via the Protobuf protocol.

Yes, there is the NetworkStream class, but unfortunately one can't use NetworkStream for UDP.

2.4.1.1 Example

```
UdpStream stream1 = new UdpStream(new UdpClient(new IPEndPoint(IPAddress.Any, 1234)));
UdpStream stream2 = new UdpStream(new UdpClient());
stream2.Client.Connect("localhost", 1234);
byte[] buffer = new byte[] { 0x01, 0x02, 0x03, 0x04 };
stream2.Write(buffer, 0, buffer.Length);
foreach (byte b in buffer)
{
    Assert.AreEqual(b, (byte)stream1.ReadByte());
}
```