# System.Memory.Data library for .NET

## Binary Data

The BinaryData type provides a lightweight abstraction for a payload of bytes. It provides convenient helper methods to get out commonly used primitives, such as streams, strings, or bytes. The assumption when converting to and from string is that the encoding is UTF-8.

### Data ownership

When using the byte[] or ReadOnlyMemory<byte> constructors or methods, BinaryData will wrap the passed in bytes. When using streams, strings, or rich model types that will be serialized as Json, the data is converted into bytes and will be maintained by BinaryData. Thus, if you are using bytes to create your instance of BinaryData, changes to the underlying data will be reflected in BinaryData as it does not copy the bytes.

### Usage

The main value of this type is its ability to easily convert from string to bytes to stream. This can greatly simplify API surface areas by exposing this type as opposed to numerous overloads or properties.

To/From string:

var data = new BinaryData("some data");

// ToString will decode the bytes using UTF-8

Console.WriteLine(data.ToString()); // prints "some data"

To/From bytes:

byte[] bytes = Encoding.UTF8.GetBytes("some data");

// Create BinaryData using a constructor ...

BinaryData data = new BinaryData(bytes);

// Or using a static factory method.

data = BinaryData.FromBytes(bytes);

// There is an implicit cast defined for ReadOnlyMemory<byte>

ReadOnlyMemory<byte> rom = data;

// There is also an implicit cast defined for ReadOnlySpan<byte>

ReadOnlySpan<byte> ros = data;

// there is also a ToMemory method that gives access to the ReadOnlyMemory.

rom = data.ToMemory();

// and a ToArray method that converts into a byte array.

byte[] array = data.ToArray();

To/From stream:

var bytes = Encoding.UTF8.GetBytes("some data");

Stream stream = new MemoryStream(bytes);

var data = BinaryData.FromStream(stream);

// Calling ToStream will give back a stream that is backed by ReadOnlyMemory, so it is not writable.

stream = data.ToStream();

Console.WriteLine(stream.CanWrite); // prints false

BinaryData also can be used to integrate with ObjectSerializer. By default, the JsonObjectSerializer will be used, but any serializer deriving from ObjectSerializer can be used.

var model = new CustomModel

{

A = "some text",

B = 5,

C = true

};

var data = BinaryData.FromObjectAsJson(model);

model = data.ToObjectFromJson<CustomModel>();