Introduction

This package provides API for data serialization/deserialization into MessagePack and JSON formats.

Supported Platforms:

- PC/Mac
- iOS
- Android
- WebGL

API

- Json
  - Serialize
  - SerializeToString
  - Deserialize
- MsgPack
  - Serialize
  - Deserialize

Installation

Nuget

```
PM> Install-Package GameDevWare.Serialization
```

Unity3D Json + MessagePack Serializer

Example

Serialize object into Stream using MessagePack serializer:

```
var outputStream = new MemoryStream();
MsgPack.Serialize(new { field1 = 1, field2 = 2 }, outputStream);
outputStream.Position = 0; // rewind stream before copying/read
```

Deserialize object from Stream using MessagePack serializer:

```
Stream inputStream;
MsgPack.Deserialize(typeof(MyObject), inputStream); -> instance of MyObject
// or
MsgPack.Deserialize<MyObject>(inputStream); -> instance of MyObject
```

Breaking Change in v2.0

Message Pack Endianness

Message Pack serialization prior to v2.0 uses little-endian byte order for multi-byte integers. That doesn't correspond to specification.
Data saved with **little-endian** formatting could be re-written to **big-endian** with following code:

```
var context = new SerializationContext { Options = SerializationOptions.SuppressTypeInformat
using (var fileStream = File.Open("<path to file>", FileMode.Open, FileAccess.ReadWrite))
{

    var reader = new MsgPackReader(fileStream, context, Endianness.LittleEndian);
    var value = reader.ReadValue(typeof(object));
    fileStream.Position = 0;
    var writer = new MsgPackWriter(fileStream, context);
    writer.WriteValue(value, typeof(object));
    fileStream.SetLength(fileStream.Position);
}
```

Data Contract Attributes

- The IgnoreDataMember attribute is only honored when used with unmarked types. This includes types that are not marked with DataContract attribute.

- You can apply the DataMember attribute to **PUBLIC** fields, and properties.

- The DataMember and IgnoreDataMember attributes are ignored if it is applied to static members.

- The DataMember attribute is ignored if DataContract attribute is not applied.

- During serialization, property-get code is called for property data members to get the value of the properties to be serialized.

- During deserialization, an new object is first created, with calling an empty constructor on the type. Then all data members are deserialized.

- During deserialization, property-set code is called for property data members to set the properties to the value being deserialized.

Mapping Types

MessagePack/Json serializer is guided by Data Contract rules.
Its behaviour can be changed with DataContract, DataMember, IgnoreDataMember attributes.

Attributes can be from *System.Runtime.Serialization.dll* or your attributes with same names.

Supported Types

- Primitives: Boolean, Byte, Double, Int16, Int32, Int64, SBytes, Single, UInt16, UInt32, UInt64, String

- Standard Types: Decimal, DateTimeOffset, DateTime, TimeSpan, Guid, Uri, Version, DictionaryEntry

- Unity3D Types: Bounds, Vector, Matrix4x4, Quaternion, Rect, Color . . .

- Binary: Stream, byte[]

- Lists: Array, ArrayList, List, HashSet and any other IEnumerable types with **Add** method.

- Maps: Hashtable, Dictionary, and other IDictionary types

- Nullable types
- Enums
- Custom classes

Custom Type Serializers

To implement a custom TypeSerializer you need to inherit it from *TypeSerializer* and override Deserialize and Serialize methods.

```
public sealed class GuidSerializer : TypeSerializer
{
    public override Type SerializedType { get { return typeof(Guid); } }

    public override object Deserialize(IJsonReader reader)
    {
        // General rule of 'Deserialize' is to leave reader on
        // last token of deserialized value. It is EndOfObject or EndOfArray, or Value.

        // 'nextToken: true' will call 'reader.NextToken()' AFTER 'ReadString()'.
        // Since it is last token on de-serialized value we set 'nextToken: false'.
        var guidStr = reader.ReadString(nextToken: false);
        var value = new Guid(guidStr);
        return value;
    }

    public override void Serialize(IJsonWriter writer, object valueObj)
    {
        var value = (Guid)valueObj; // valueObj is not null
        var guidStr = value.ToString();
        writer.Write(guidStr);
    }
}
```

Then you need to register your class in *Json.DefaultSerializers* collection or mark it with TypeSerializerAttribute.

Extra Type Information

There is additional type information with each serialized object. It increases size of the serialized data. If you do not want to store object's type information, specify *SuppressTypeInformation* when calling **Serialize** method.

```
MsgPack.Serialize(value, stream, SerializationOptions.SuppressTypeInformation);
```

If you want to ignore type information when deserializing an object, specify *SuppressTypeInformation* when calling **Deserialize** method.

```
MsgPack.Deserialize(typeof(MyObject), stream, SerializationOptions.SuppressTypeInformation)
```

Contacts

Please send any questions at support@gamedevware.com

License

MIT