Simple Data Compression

Introduction

Simple Data Compression is a Unity plugin that offers a simplified API for saving and loading compressed data in the form of binary files.

With **Simple Data Compression** you can compress and save your data with a single line of code. Similarly, you can load and decompress data with a single line. Alternatively, use **Simple Data Compression** to convert your data into compressed byte arrays and build your own custom data containers.

Many different 1, 2, 4, and 8-byte data types are supported:

- 1-byte: *byte, sbyte, bool*
- 2-byte: *short, ushort, char*
- 4-byte: *int, uint, float*
- 8-byte: *long, ulong*

Simple Data Compression comes with the following demo scenes:

- **Data Compression:** Build a sample 2D float array which is compressed and saved with a single command. Load and decompress the array, check for errors, and compare the file size with an uncompressed binary file.
- **Text Compression:** Load a string consisting of the play *Hamlet*, compress and save the array with a single line of code. Load and decompress the array and compare the file size with the uncompressed file.
- **Data Container Compression:** Build a custom class consisting of a several data types (float, ushort, bool, ect.) and convert each data type into a compressed byte array which is saved as a binary file. Decompress this file, and check each data type for errors.

TinyAngle Labs

Contents

Quick Start Guide

Compressing and Saving Data

For the purposes of this guide, assume you are starting with a 2D float array called *dataArray* of size 2000x1000. To compress and save this data as a binary file in the default *Applications.persistentDataPath* use the following command:

```
SDCompression.SaveCompressed(dataArray, “Compressed_Data”);
```

This will create the compressed data file *Compressed_data.dat* in the *Applications.persistentDataPath* directory. See the function definition for how to specify a different directory.

To load this file, decompress it, and retrieving the float array, use the following command:

```
float[,] loadedDataArray = SDCompression.LoadCompressed(2000, 1000, “Compressed_Data”);
```

Note that the array size must be specified when loading the compressed data.

Classes and Functions

SDCompression

```
public static class SDCompression
```

Functions:

```
public static bool SaveCompressed<T>(T[] data, string filename, string dir = “”)
public static bool SaveCompressed<T>(T[,] data, string filename, string dir = “”)
```

Compress 1D or 2D data array of type T and save as “[filename].dat”. “dir” is an optional parameter for the save directory. If left blank, the file will be saved in “Application.persistentDataPath”. Returns “true” is operation was successful.

```
public static T[] LoadCompressed<T>(
             int size,
             string filename,
             string dir = “”)
public static T[,] LoadCompressed<T>(
             int sizeA,
             int SizeB,
             string filename,
             string dir = “”)
```

Load file “[filename].dat” from directory “dir” (if left blank, load directory will default to Application.persistentDataPath), decompress data, and return 1D or 2D array of type T.

```
public static byte[] CompressArrayByte<T>(T[] data, int size)
public static byte[] CompressArrayByte<T>(T[,] data, int sizeA, int sizeB)
```

Compress 1-byte data array of type T (1D or 2D) into a byte array.

```
public static ShortByteArray CompressArrayShort<T>(T[] data, int size)
public static ShortByteArray CompressArrayShort<T>(T[,] data, int sizeA, int sizeB)
```

Compress 2-byte data array of type T (1D or 2D) into a ShortByteArray.

```
public static SingleByteArray CompressArraySingle<T>(T[] data, int size)
public static SingleByteArray CompressArraySingle<T>(T[,] data, int sizeA, int sizeB)
```

Compress 4-byte data array of type T (1D or 2D) into a SingleByteArray.

```
public static LongByteArray CompressArrayLong<T>(T[] data, int size)
public static LongByteArray CompressArrayLong<T>(T[,] data, int sizeA, int sizeB)
```

Compress 8-byte data array of type T (1D or 2D) into a LongByteArray.

```
public static T[] DecompressArray<T>(
             byte[] compressedArray,
             int size)
public static T[,] DecompressArray<T>(
             byte[,] compressedArray,
             int sizeA,
             int sizeB)

public static T[] DecompressArray<T>(
             ShortByteArray compressedArray,
             int size)
public static T[,] DecompressArray<T>(
             ShortByteArray[,] compressedArray,
             int sizeA,
             int sizeB)

public static T[] DecompressArray<T>(
             SingleByteArray compressedArray,
             int size)
public static T[,] DecompressArray<T>(
             SingleByteArray[,] compressedArray,
             int sizeA,
             int sizeB)

public static T[] DecompressArray<T>(
             LongByteArray compressedArray,
             int size)
public static T[,] DecompressArray<T>(
             LongByteArray[,] compressedArray,
             int sizeA,
             int sizeB)
```
Decompress data into a 1D or 2D array of type T.

SDCompression.ShortByteArray

```
public class SDCompression.ShortByteArray
```

Storage class for 2-byte data types

Key Public Properties:

```
public byte[] byte0;
public byte[] byte1;
```

Public Methods:

Constructors:

```
public ShortByteArray(int size) // Create for data with specific size
public ShortByteArray() // Create empty container
```

SDCompression.SingleByteArray

```
public class SDCompression.SingleByteArray
```

Storage class for 4-byte data types

Key Public Properties:

```
public byte[] byte0;
public byte[] byte1;
public byte[] byte2;
public byte[] byte3;
```

Public Methods:

Constructors:

```
public SingleByteArray(int size) // Create for data with specific size
public SingleByteArray() // Create empty container
```

SDCompression.LongByteArray

```
public class SDCompression.LongByteArray
```

Storage class for 8-byte data types

Key Public Properties:

```
public byte[] byte0;
public byte[] byte1;
public byte[] byte2;
public byte[] byte3;
public byte[] byte4;
public byte[] byte5;
public byte[] byte6;
public byte[] byte7;
```

Public Methods:

Constructors:

```
public LongByteArray(int size) // Create for data with specific size
public LongByteArray() // Create empty container
```

Support

If you are having issues with Noedify or have a suggestion, please go to https://www.TinyAngleLabs.com/contact-us and get in touch.