AshLib Reference Documentation

Documentation for AshLib v2.1.0, by Dumbelfo.
https://github.com/Dumbelfo08/AshLib

Index

Contents

Structs

Color3

Used for holding the values of a RGB color

Constructors

Color3(byte, byte, byte)

```
1. public Color3(byte r, byte g, byte b)
```

Initializes a new color taking in arguments for red, green, and blue channels.

Color3(string)

```
1. public Color3(string hex)
```

Initializes a new color taking in argument for the color in hex format. As an example of valid strings are:

```
“ffffff”, “#020202”, “#eFE02Dd”
```

Fields

R

```
1. public byte R;
```

Red channel of the color. Ranges 0-255

G

```
1. public byte G;
```

Green channel of the color. Ranges 0-255

B

```
1. public byte B;
```

Blue channel of the color. Ranges 0-255

Methods

Equals

```
1. public override bool Equals (object obj)
```

Checks if two Color3 are equals

ToString

```
1. public override string ToString ()
```

Returns the color data in the format:

```
“#RRGGBB”
```

Operators

Operator == (Color3, Color3)

```
1. public static bool operator == (Color3 a, Color3 b)
```

Checks if two Color3 are equals

Operator != (Color3, Color3)

```
1. public static bool operator != (Color3 a, Color3 b)
```

Checks if two Color3 are not equals

Implicit Operator Color (Color3)

```
1. public static implicit operator System.Drawing.Color (Color3 c)
```

Casts into a System.Drawing.Color

Implicit Operator Color3 (Color)

```
1. public static implicit operator Color3(System.Drawing.Color c)
```

Casts into a Color3

Vec2

Struct for holding two floats, components of a bidimensional vector

Constructors

Vec2(float, float)

```
1. public Vec2 (float x, float y)
```

Initializes with x and y component

Fields

X

```
1. public float X;
```

X component of the Vector

Y

```
1. public float Y;
```

Y component of the Vector

Methods

Equals

```
1. public override bool Equals (object obj)
```

Checks if two Vec2 contain the same data

ToString

```
1. public override string ToString ()
```

Returns the vector components in the format:

```
“(X, Y)”
```

Operators

Operator == (Vec2, Vec2)

```
1. public static bool operator == (Vec2 a, Vec2 b)
```

Checks if two Vec2 contain the same data

Operator != (Vec2, Vec2)

```
1. public static bool operator != (Vec2 a, Vec2 b)
```

Checks if two Vec2 do not contain the same data

Vec3

Struct for holding three floats, components of a tridimensional vector

Constructors

Vec3(float, float, float)

```
1. public Vec3 (float x, float y, float z)
```

Initializes with x, y and z component

Fields

X

```
1. public float X;
```

X component of the Vector

Y

```
1. public float Y;
```

Y component of the Vector

Z

```
1. public float Z;
```

Z component of the Vector

Methods

Equals

```
1. public override bool Equals (object obj)
```

Checks if two Vec3 contain the same data

ToString

```
1. public override string ToString ()
```

Returns the vector components in the format:

```
“(X, Y, Z)”
```

Operators

Operator == (Vec3, Vec3)

```
1. public static bool operator == (Vec3 a, Vec3 b)
```

Checks if two Vec3 contain the same data

Operator != (Vec3, Vec3)

```
1. public static bool operator != (Vec3 a, Vec3 b)
```

Checks if two Vec3 do not contain the same data

Vec4

Struct for holding four floats, components of a four-dimensional vector

Constructors

Vec4(float, float, float, float)

```
1. public Vec4 (float x, float y, float z, float w)
```

Initializes with x, y, z and w component

Fields

X

```
1. public float X;
```

X component of the Vector

Y

```
1. public float Y;
```

Y component of the Vector

Z

```
1. public float Z;
```

Z component of the Vector

W

```
1. public float W;
```

W component of the Vector

Methods

Equals

```
1. public override bool Equals (object obj)
```

Checks if two Vec4 contain the same data

ToString

```
1. public override string ToString ()
```

Returns the vector components in the format:

```
“(X, Y, Z, W)”
```

Operators

Operator == (Vec4, Vec4)

```
1. public static bool operator == (Vec4 a, Vec4 b)
```

Checks if two Vec4 contain the same data

Operator != (Vec4, Vec4)

```
1. public static bool operator != (Vec4 a, Vec4 b)
```

Checks if two Vec4 do not contain the same data

Date

Represents a time between the year 1488 and 2511, down to seconds. It's most important characteristic is CPTF(Compressed printable date format), which will transform the date into a 6 characters long string in base64

Constructors

Date(byte, byte, byte, byte, byte, ushort)

```
1. public Date(byte s, byte m, byte h, byte d, byte mo, ushort y)
```

Initializes the Date with seconds, minutes, hours, days, months, and years(in that order). Numbers will be cropped (seconds 0-59...)

Date(string)

```
1. public Date(string cptf)
```

Initializes the Date directly from CPTF format.

Date()

```
1. public Date()
```

Do not use. Initializes an invalid Date. Is only for internal purposes, but structs can't have private constructors without arguments.

Fields

Invalid

```
1. public static readonly Date Invalid;
```

Invalid Date instance. Use better than the constructor. Primarily for internal use.

Properties

seconds

```
1. public byte seconds {get;}
```

The seconds of the Date(0-59)

minutes

```
1. public byte minutes {get;}
```

The minutes of the Date(0-59)

hours

```
1. public byte hours {get;}
```

The hours of the Date(0-23)

days

```
1. public byte days {get;}
```

The day of the Date(1-31)

months

```
1. public byte months {get;}
```

The month of the Date(1-12)

years

```
1. public ushort years {get;}
```

The year of the Date(1488-2511)

Methods

ToCPTF

```
1. public string ToCPTF ()
```

Transforms the Date instance into the CPTF format

FromCPTF

```
1. public static Date FromCPTF (string cptf)
```

Transforms a date in CPTF format into an instance. Does the same as the constructor

Equals

```
1. public override bool Equals (object obj)
```

Checks if two Dates are the same

ToString

```
1. public override string ToString ()
```

Returns the Date in the format:

```
“Day/Month/Year Hour:Minute:Second”
```

Operators

Operator == (Date, Date)

```
1. public static bool operator == (Date a, Date b)
```

Checks if two Dates are equals

Operator != (Date, Date)

```
1. public static bool operator != (Date a, Date b)
```

Checks if two Dates are not equals

Explicit Operator DateTime (Date)

```
1. public static explicit operator DateTime (Date d)
```

Casts to DateTime

Explicit Operator Date (DateTime)

```
1. public static explicit operator Date (DateTime d)
```

Casts to Date from DateTime

Enums

Type

Represents the type of value in an AshFile camp.

Fields

Invalid

```
1. Invalid = -1;
```

Invalid type

ByteArray

```
1. ByteArray = 0;
```

Array of bytes type

String

```
1. String = 1;
```

String type

Byte

```
1. Byte = 2;
```

Byte type. 1 byte unsigned integer

Ushort

```
1. Ushort = 3;
```

Unsigned short type. 2 byte unsigned integer

Uint

```
1. Uint = 4;
```

Unsigned int type. 4 byte unsigned integer

Ulong

```
1. Ulong = 5;
```

Unsigned long type. 8 byte unsigned integer

Sbyte

```
1. Sbyte = 6;
```

Signed byte type. 1 byte signed integer

Short

```
1. Short = 7;
```

Signed short type. 2 byte signed integer

Int

```
1. Int = 8;
```

Signed int type. 4 byte signed integer

Long

```
1. Long = 9;
```

Signed long type. 8 byte signed integer

Color

```
1. Color = 10;
```

Color3 type. Struct defined earlier

Float

```
1. Float = 11;
```

Float type. 4 byte floating point number

Double

```
1. Double = 12;
```

Double type. 8 byte floating point number

Vec2

```
1. Vec2 = 13;
```

Vec2 type. 2 component vector, struct defined earlier

Vec3

```
1. Vec3 = 14;
```

Vec3 type. 3 component vector, struct defined earlier

Vec4

```
1. Vec4 = 15;
```

Vec4 type. 4 component vector, struct defined earlier

Bool

```
1. Bool = 16;
```

Boolean type

UbyteArray

```
1. UbyteArray = 17;
```

Array of bytes type. 1 byte unsigned integer array

UshortArray

```
1. UshortArray = 18;
```

Array of ushorts type. 2 byte unsigned integer array

UintArray

```
1. UintArray = 19;
```

Array of uints type. 4 byte unsigned integer array

UlongArray

```
1. UshortArray = 20;
```

Array of ulongs type. 4 byte unsigned integer array

SbyteArray

```
1. SbyteArray = 21;
```

Array of sbytes type. 1 byte signed integer array

ShortArray

```
1. ShortArray = 22;
```

Array of shorts type. 2 byte signed integer array

IntArray

```
1. IntArray = 23;
```

Array of ints type. 4 byte signed integer array

LongArray

```
1. LongArray = 24;
```

Array of longs type. 8 byte signed integer array

FloatArray

```
1. FloatArray = 25;
```

Array of floats type. 4 byte floating point number array

DoubleArray

```
1. DoubleArray = 26;
```

Array of doubles type. 8 byte floating point number array

Date

```
1. Date = 27;
```

Date type. Struct defined earlier

Classes

DeltaHelper

Class used for easily calculating fps and deltaTime

Properties

deltaTime

```
1. public double deltaTime {get;}
```

Time that passes between frames, in milliseconds

fps

```
1. public double fps {get;}
```

Frames per second. Is calculated each frame, and so it fluctuates rapidly

stableFps

```
1. public double stableFps {get;}
```

Frames per second. Will update once every second(default behavior, can be changed). Thought for displaying current fps

Methods

Start

```
1. public void Start ()
```

Has to be called to start the utility. Will initialize the internal clock. Call it once at the start of the application

Frame

```
1. public void Frame ()
```

Call it at the very end of each frame, every frame. Updates all the values

Target

```
1. public void Target (double FPS)
```

Call it at the end of each frame, but before Frame(), to achieve the desired fps(the argument)

SetStableUpdateTime

```
1. public void SetStableUpdateTime (double milliseconds)
```

Changes how often will the stableFps update

GetTime

```
1. public double GetTime ()
```

Returns the whole time since start, in seconds

Dependencies

Used for handling files in a central folder of the application. Reminiscent of ".minecraft" folder

Constructors

Dependencies(string, bool, string[], string[])

```
1. public Dependencies (string path, bool config, string[] directories, string[]
files)
```

Initializes the utility. The first argument is the main path, the second is if you want the config AshFile to be created, the directories array specifies folders inside the main path that will be created, and the files array specifies files that will be created inside the main path or subfolders, will be created empty.

Fields

path

```
1. public string path;
```

The master path. For example:

```
“C://Users/user22/AppData/Roaming/HelloWorld”
```

config

```
1. public AshFile config;
```

The main configuration, in an AshFile. The path to it is “mainpath/config.ash”

Methods

ReadFileText

```
1. public string ReadFileText (string p)
```

Will read the contents of a file as text of the file in masterpath + argument

ReadAshFile

```
1. public AshFile ReadAshFile (string p)
```

Will read the contents of a file as an AshFile of the file in masterpath + argument

SaveFileText

```
1. public void SaveFileText (string p, string t)
```

Will save the text(argument t) in a file in masterpath + argument p

SaveAshFile

```
1. public void SaveAshFile (string p, AshFile a)
```

Will save the text(argument t) in a file in masterpath + argument p

CreateDir

```
1. public void CreateDir (string p)
```

Will create a new directory in masterpath + argument

CampValue

Holds the value of an AshFile camp. This value can be of different types(string, numbers...)

Constructors

CampValue(byte[])

```
1. public CampValue (byte[] p)
```

Initializes the value as a byte array, with the value being the argument

CampValue(string)

```
1. public CampValue (string p)
```

Initializes the value as a string, with the value being the argument

CampValue(byte)

```
1. public CampValue (byte p)
```

Initializes the value as an unsigned byte, with the value being the argument

CampValue(ushort)

```
1. public CampValue (ushort p)
```

Initializes the value as an unsigned short, with the value being the argument

CampValue(uint)

```
1. public CampValue (uint p)
```

Initializes the value as an unsigned int, with the value being the argument

CampValue(ulong)

```
1. public CampValue (ulong p)
```

Initializes the value as an unsigned long, with the value being the argument

CampValue(sbyte)

```
1. public CampValue (sbyte p)
```

Initializes the value as a signed byte, with the value being the argument

CampValue(short)

```
1. public CampValue (sbyte p)
```

Initializes the value as a signed byte, with the value being the argument

CampValue(int)

```
1. public CampValue (int p)
```

Initializes the value as a signed int, with the value being the argument

CampValue(long)

```
1. public CampValue (long p)
```

Initializes the value as a signed long, with the value being the argument

CampValue(Color3)

```
1. public CampValue (Color3 p)
```

Initializes the value as a Color3 (struct defined earlier), with the value being the argument

CampValue(float)

```
1. public CampValue (float p)
```

Initializes the value as a float, with the value being the argument

CampValue(double)

```
1. public CampValue (double p)
```

Initializes the value as a double, with the value being the argument

CampValue(Vec2)

```
1. public CampValue (Vec2 p)
```

Initializes the value as a Vec2 (struct defined earlier), with the value being the argument

CampValue(Vec3)

```
1. public CampValue (Vec3 p)
```

Initializes the value as a Vec3 (struct defined earlier), with the value being the argument

CampValue(Vec4)

```
1. public CampValue (Vec4 p)
```

Initializes the value as a Vec4 (struct defined earlier), with the value being the argument

CampValue(bool)

```
1. public CampValue (bool p)
```

Initializes the value as a boolean, with the value being the argument

CampValue(ushort[])

```
1. public CampValue (ushort[] p)
```

Initializes the value as an unsigned short array, with the value being the argument

CampValue(uint[])

```
1. public CampValue (uint[] p)
```

Initializes the value as an unsigned int array, with the value being the argument

CampValue(ulong[])

```
1. public CampValue (ulong[] p)
```

Initializes the value as an unsigned short array, with the value being the argument

CampValue(sbyte[])

```
1. public CampValue (sbyte[] p)
```

Initializes the value as a signed byte array, with the value being the argument

CampValue(short[])

```
1. public CampValue (short[] p)
```

Initializes the value as a signed short array, with the value being the argument

CampValue(int[])

```
1. public CampValue (int[] p)
```

Initializes the value as a signed int array, with the value being the argument

CampValue(long[])

```
1. public CampValue (long[] p)
```

Initializes the value as a signed long array, with the value being the argument

CampValue(float[])

```
1. public CampValue (float[] p)
```

Initializes the value as a float array, with the value being the argument

CampValue(double[])

```
1. public CampValue (double[] p)
```

Initializes the value as a double array, with the value being the argument

CampValue(Date)

```
1. public CampValue (Date p)
```

Initializes the value as a Date (struct defined earlier), with the value being the argument

Fields

Null

```
1. public static readonly CampValue Null;
```

A null instance, mostly for internal use, for errors and so.

Properties

type

```
1. public Type type {get;}
```

Gets the type of the value represented, in one of the structs defined earlier

Methods

GetValue

```
1. public object GetValue ()
```

Returns the value, no matter its type.

CanGetByteArray

```
1. public bool CanGetByteArray (out byte[] p)
```

If the CampValue's value is of the type ByteArray, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetString

```
1. public bool CanGetString (out string p)
```

If the CampValue's value is of the type String, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetByte

```
1. public bool CanGetByte (out byte p)
```

If the CampValue's value is of the type Byte, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetUshort

```
1. public bool CanGetUshort (out ushort p)
```

If the CampValue's value is of the type Ushort it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetUint

```
1. public bool CanGetUint (out uint p)
```

If the CampValue's value is of the type Uint, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetLong

```
1. public bool CanGetLong (out long p)
```

If the CampValue's value is of the type Ulong, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetSbyte

```
1. public bool CanGetSbyte (out sbyte p)
```

If the CampValue's value is of the type Sbyte, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetShort

```
1. public bool CanGetShort (out short p)
```

If the CampValue's value is of the type Short, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetInt

```
1. public bool CanGetInt (out int p)
```

If the CampValue's value is of the type Int, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetLong

```
1. public bool CanGetLong (out long p)
```

If the CampValue's value is of the type Long, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetColor3

```
1. public bool CanGetColor3 (out Color3 p)
```

If the CampValue's value is of the type Color, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetFloat

```
1. public bool CanGetFloat (out float p)
```

If the CampValue's value is of the type Float, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetDouble

```
1. public bool CanGetDouble (out double p)
```

If the CampValue's value is of the type Double, it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetVec2

```
1. public bool CanGetVec2 (out Vec2 p)
```

If the CampValue's value is of the type Vec2 it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetVec3

```
1. public bool CanGetVec3 (out Vec3 p)
```

If the CampValue's value is of the type Vec3 it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetVec4

```
1. public bool CanGetVec4 (out Vec4 p)
```

If the CampValue's value is of the type Vec4 it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetBool

```
1. public bool CanGetBool (out bool p)
```

If the CampValue's value is of the type Bool it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetUbyteArray

```
1. public bool CanGetUbyteArray (out byte[] p)
```

If the CampValue's value is of the type UbyteArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetUshortArray

```
1. public bool CanGetUshortArray (out ushort[] p)
```

If the CampValue's value is of the type UshortArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetUintArray

```
1. public bool CanGetUintArray (out uint[] p)
```

If the CampValue's value is of the type UintArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetUlongArray

```
1. public bool CanGetUlongArray (out ulong[] p)
```

If the CampValue's value is of the type UlongArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetSbyteArray

```
1. public bool CanGetSbyteArray (out sbyte[] p)
```

If the CampValue's value is of the type SbyteArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetShortArray

```
1. public bool CanGetShortArray (out short[] p)
```

If the CampValue's value is of the type ShortArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetIntArray

```
1. public bool CanGetIntArray (out int[] p)
```

If the CampValue's value is of the type IntArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetLongArray

```
1. public bool CanGetLongArray (out long[] p)
```

If the CampValue's value is of the type LongArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetFloatArray

```
1. public bool CanGetFloatArray (out float[] p)
```

If the CampValue's value is of the type FloatArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetDoubleArray

```
1. public bool CanGetDoubleArray (out double[] p)
```

If the CampValue's value is of the type DoubleArray it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

CanGetDate

```
1. public bool CanGetDate (out Date p)
```

If the CampValue's value is of the type Date it will return true and output the value on the out argument, if not, it will return false and output null in the argument.

Equals

```
1. public override bool Equals (object obj)
```

Checks if two CampValues are the same

ToString

```
1. public override string ToString ()
```

Returns the vale, or if the value is an array, returns all its components to string.

Operators

Operator == (CampValue, CampValue)

```
1. public static bool operator == (CampValue a, CampValue b)
```

Checks if two CampValue's values are equals

Operator != (CampValue, CampValue)

```
1. public static bool operator != (CampValue a, CampValue b)
```

Checks if two CampValue's values are not equals

AshFileException

Used internally for all exceptions that happen surrounding AshFiles

```
1. [Serializable]
2. internal class AshFileException : Exception
```

Properties

errorCode

```
1. public int errorCode {get;}
```

Different errors give different codes

Methods

GetFullMessage

```
1. public string GetFullMessage ()
```

Returns the whole information needed about the exception(error code, message, stack trace...) in a nice format

ToString

```
1. public override string ToString ()
```

Returns the same as GetFullMessage

GetObjectData

```
1. public override void GetObjectData(SerializationInfo info, StreamingContext
context)
```

Does something about Serialization. Honestly no idea

AshFile

A Data structure made up from camps, that each have a name and a value(represented by a CampValue object because it can be of many types). This structure makes it easy to be saved into a file in a new file format (.ash)

Constructors

AshFile(Dictionary<string, CampValue>)

```
1. public AshFile (Dictionary<string, CampValue> d)
```

Initializes a new AshFile using an existing Dictionary, the structure used internally

AshFile(string)

```
1. public AshFile(string path)
```

Loads an AshFile from the specified path

AshFile()

```
1. public AshFile()
```

Initializes a new empty AshFile

Fields

path

```
1. public string? path;
```

The path of the file, if it has any

format

```
1. public byte format;
```

Specifies the format in which the file will be saved

Properties

data

```
1. public Dictionary<string, CampValue> data {get;}
```

The actual structure holding the data

Methods

AsString

```
1. public string AsString ()
```

Produces a list of all the elements, in the format:

```
“Name1: Value1
 Name2: Value2”
```

Load(string)

```
1. public void Load (string path)
```

Loads the file from the path and save the path internally

Load()

```
1. public void Load ()
```

Loads the file from the path saved internally

Save(string)

```
1. public void Save (string path)
```

Saves the file from the path and saves the path internally

Save()

```
1. public void Save ()
```

Saves the file from the internal path

ReadFromFile

```
1. public static Dictionary<string, CampValue> ReadFromFile(string path, out byte f)
```

Reads the file from the path argument and returns it. Outputs the format in the f argument

ReadFromBytes

```
1. public static Dictionary<string, CampValue> ReadFromFile(byte[] fileBytes, out
byte f)
```

Reads the file from the byte array and returns it. Outputs the format in the f argument

WriteToFile

```
1. public static void WriteToFile (string path, Dictionary<string, CampValue>
dictionary, byte format)
```

Writes the data into a file in the path argument. Format can be specified

WriteToBytes(Dictionary<string, CampValue>, byte)

```
1. public static byte[] WriteToBytes (Dictionary<string, CampValue> dictionary, byte
format)
```

Transforms the data into the byte representation. Format can be specified

WriteToBytes()

```
1. public byte[] WriteToBytes ()
```

Transforms the current instance into the byte representation.

GetErrorCount

```
1. public static ulong GetErrorCount ()
```

Gets the number of errors that occurred. These errors occur while converting to/from file

GetErrorLog

```
1. public static string GetErrorLog ()
```

Gets the whole log of errors that occurred, nicely formatted. These errors occur while converting to/from file

EmptyErrors

```
1. public static void EmptyErrors ()
```

Empties all the errors and sets the count to zero.

DeepCopy

```
1. public static AshFile DeepCopy (AshFile a)
```

Copies all the camps from one AshFile to a new one, keeping the references different

ExistsCamp

```
1. public bool ExistsCamp (string name)
```

Checks if a specific camp exists, returns true if it does.

SetCamp(string, CampValue)

```
1. public void SetCamp (string name, CampValue val)
```

Sets the camp with the argument name to the CampValue(this CampValue can be of any type)

SetCamp(string, byte[])

```
1. public void SetCamp (string name, byte[] val)
```

Sets the camp with the argument name to the byte array

SetCamp(string, string)

```
1. public void SetCamp (string name, string val)
```

Sets the camp with the argument name to the string

SetCamp(string, byte)

```
1. public void SetCamp (string name, byte val)
```

Sets the camp with the argument name to the byte

SetCamp(string, ushort)

```
1. public void SetCamp (string name, ushort val)
```

Sets the camp with the argument name to the ushort

SetCamp(string, uint)

```
1. public void SetCamp (string name, uint val)
```

Sets the camp with the argument name to the uint

SetCamp(string, ulong)

```
1. public void SetCamp (string name, ulong val)
```

Sets the camp with the argument name to the ulong

SetCamp(string, sbyte)

```
1. public void SetCamp (string name, sbyte val)
```

Sets the camp with the argument name to the sbyte

SetCamp(string, short)

```
1. public void SetCamp (string name, short val)
```

Sets the camp with the argument name to the short

SetCamp(string, int)

```
1. public void SetCamp (string name, int val)
```

Sets the camp with the argument name to the int

SetCamp(string, long)

```
1. public void SetCamp (string name, long val)
```

Sets the camp with the argument name to the long

SetCamp(string, Color3)

```
1. public void SetCamp (string name, Color3 val)
```

Sets the camp with the argument name to the Color3(struct defined earlier)

SetCamp(string, float)

```
1. public void SetCamp (string name, float val)
```

Sets the camp with the argument name to the float

SetCamp(string, double)

```
1. public void SetCamp (string name, double val)
```

Sets the camp with the argument name to the double

SetCamp(string, Vec2)

```
1. public void SetCamp (string name, Vec2 val)
```

Sets the camp with the argument name to the Vec2(struct defined earlier)

SetCamp(string, Vec3)

```
1. public void SetCamp (string name, Vec3 val)
```

Sets the camp with the argument name to the Vec3(struct defined earlier)

SetCamp(string, Vec4)

```
1. public void SetCamp (string name, Vec4 val)
```

Sets the camp with the argument name to the Vec4(struct defined earlier)

SetCamp(string, bool)

```
1. public void SetCamp (string name, bool val)
```

Sets the camp with the argument name to the bool

SetCamp(string, ushort[])

```
1. public void SetCamp (string name, ushort[] val)
```

Sets the camp with the argument name to the ushort array

SetCamp(string, uint[])

```
1. public void SetCamp (string name, uint[] val)
```

Sets the camp with the argument name to the uint array

SetCamp(string, ulong[])

```
1. public void SetCamp (string name, ulong[] val)
```

Sets the camp with the argument name to the ulong array

SetCamp(string, sbyte[])

```
1. public void SetCamp (string name, sbyte[] val)
```

Sets the camp with the argument name to the sbyte array

SetCamp(string, short[])

```
1. public void SetCamp (string name, short[] val)
```

Sets the camp with the argument name to the short array

SetCamp(string, int[])

```
1. public void SetCamp (string name, int[] val)
```

Sets the camp with the argument name to the int array

SetCamp(string, long[])

```
1. public void SetCamp (string name, long[] val)
```

Sets the camp with the argument name to the long array

SetCamp(string, float[])

```
1. public void SetCamp (string name, float[] val)
```

Sets the camp with the argument name to the float array

SetCamp(string, double[])

```
1. public void SetCamp (string name, double[] val)
```

Sets the camp with the argument name to the double array

SetCamp(string, Date)

```
1. public void SetCamp (string name, Date val)
```

Sets the camp with the argument name to the Date(struct defined earlier)

InitializeCamp(string, CampValue)

```
1. public void InitializeCamp (string name, CampValue val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that CampValue(this CampValue can be of any type)

InitializeCamp(string, byte[])

```
1. public void InitializeCamp (string name, byte[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that byte array

InitializeCamp(string, string)

```
1. public void InitializeCamp (string name, string val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that string

InitializeCamp(string, byte)

```
1. public void InitializeCamp (string name, byte val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that byte

InitializeCamp(string, ushort)

```
1. public void InitializeCamp (string name, ushort val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that ushort

InitializeCamp(string, uint)

```
1. public void InitializeCamp (string name, uint val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that uint

InitializeCamp(string, ulong)

```
1. public void InitializeCamp (string name, ulong val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that ulong

InitializeCamp(string, sbyte)

```
1. public void InitializeCamp (string name, sbyte val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that sbyte

InitializeCamp(string, short)

```
1. public void InitializeCamp (string name, short val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that short

InitializeCamp(string, int)

```
1. public void InitializeCamp (string name, int val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that int

InitializeCamp(string, long)

```
1. public void InitializeCamp (string name, long val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that long

InitializeCamp(string, Color3)

```
1. public void InitializeCamp (string name, Color3 val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that Color3(struct defined earlier)

InitializeCamp(string, float)

```
1. public void InitializeCamp (string name, float val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that float

InitializeCamp(string, double)

```
1. public void InitializeCamp (string name, double val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that double

InitializeCamp(string, Vec2)

```
1. public void InitializeCamp (string name, Vec2 val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that Vec2(struct defined earlier)

InitializeCamp(string, Vec3)

```
1. public void InitializeCamp (string name, Vec3 val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that Vec3(struct defined earlier)

InitializeCamp(string, Vec4)

```
1. public void InitializeCamp (string name, Vec4 val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that Vec4(struct defined earlier)

InitializeCamp(string, bool)

```
1. public void InitializeCamp (string name, bool val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that bool

InitializeCamp(string, ushort[])

```
1. public void InitializeCamp (string name, ushort[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that ushort array

InitializeCamp(string, uint[])

```
1. public void InitializeCamp (string name, uint[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that uint array

InitializeCamp(string, ulong[])

```
1. public void InitializeCamp (string name, ulong[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that ulong array

InitializeCamp(string, sbyte[])

```
1. public void InitializeCamp (string name, sbyte[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that sbyte array

InitializeCamp(string, short[])

```
1. public void InitializeCamp (string name, short[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that short array

InitializeCamp(string, int[])

```
1. public void InitializeCamp (string name, int[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that int array

InitializeCamp(string, long[])

```
1. public void InitializeCamp (string name, long[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that long array

InitializeCamp(string, float[])

```
1. public void InitializeCamp (string name, float[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that float array

InitializeCamp(string, double[])

```
1. public void InitializeCamp (string name, double[] val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that double array

InitializeCamp(string, Date)

```
1. public void InitializeCamp (string name, Date val)
```

If the camp with that name exists, nothing will happen. If the camp with that name doesn't exists, it will set it to that Date(struct defined earlier)

GetCampValue

```
1. public CampValue GetCampValue (string name)
```

Returns the value of a camp. If the camp doesn't exists, a null CampValue will be returned

CanGetCampValue

```
1. public bool CanGetCampValue (string name, out CampValue val)
```

If the camp doesn't exists, returns false. If it is possible to get the value, it will return true and output the value in the out argument

GetCamp

```
1. public object GetCamp (string name)
```

Returns the value of a camp, directly as an object. If the camp doesn't exists, a null object will be returned

CanGetCamp

```
1. public bool CanGetCamp (string name, out object val)
```

If the camp doesn't exists, returns false. If it is possible to get the value, it will return true and output the value directly as an object in the out argument

CanGetCampAsByteArray

```
1. public bool CanGetCampAsByteArray (string name, out byte[] val)
```

If the camp doesn't exists or isn't of the type ByteArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsString

```
1. public bool CanGetCampAsString (string name, out string val)
```

If the camp doesn't exists or isn't of the type String, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsByte

```
1. public bool CanGetCampAsByte (string name, out byte val)
```

If the camp doesn't exists or isn't of the type Byte, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsUshort

```
1. public bool CanGetCampAsUshort (string name, out ushort val)
```

If the camp doesn't exists or isn't of the type Ushort, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsUint

```
1. public bool CanGetCampAsUint (string name, out uint val)
```

If the camp doesn't exists or isn't of the type Uint, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsUlong

```
1. public bool CanGetCampAsUlong (string name, out ulong val)
```

If the camp doesn't exists or isn't of the type Ulong, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsSbyte

```
1. public bool CanGetCampAsSbyte (string name, out sbyte val)
```

If the camp doesn't exists or isn't of the type Sbyte, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsShort

```
1. public bool CanGetCampAsShort (string name, out short val)
```

If the camp doesn't exists or isn't of the type Short, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsInt

```
1. public bool CanGetCampAsInt (string name, out int val)
```

If the camp doesn't exists or isn't of the type Int, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsLong

```
1. public bool CanGetCampAsLong (string name, out long val)
```

If the camp doesn't exists or isn't of the type Long, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsColor

```
1. public bool CanGetCampAsColor (string name, out Color3 val)
```

If the camp doesn't exists or isn't of the type Color, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsFloat

```
1. public bool CanGetCampAsFloat (string name, out float val)
```

If the camp doesn't exists or isn't of the type Float, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsDouble

```
1. public bool CanGetCampAsDouble (string name, out double val)
```

If the camp doesn't exists or isn't of the type Double, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsVec2

```
1. public bool CanGetCampAsVec2 (string name, out Vec2 val)
```

If the camp doesn't exists or isn't of the type Vec2, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsVec3

```
1. public bool CanGetCampAsVec3 (string name, out Vec3 val)
```

If the camp doesn't exists or isn't of the type Vec3 it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsVec4

```
1. public bool CanGetCampAsVec4 (string name, out Vec4 val)
```

If the camp doesn't exists or isn't of the type Vec4, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsBool

```
1. public bool CanGetCampAsBool (string name, out bool val)
```

If the camp doesn't exists or isn't of the type Bool, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsUbyteArray

```
1. public bool CanGetCampAsUbyteArray (string name, out byte[] val)
```

If the camp doesn't exists or isn't of the type UbyteArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsUshortArray

```
1. public bool CanGetCampAsUshortArray (string name, out ushort[] val)
```

If the camp doesn't exists or isn't of the type UshortArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsUintArray

```
1. public bool CanGetCampAsUintArray (string name, out uint[] val)
```

If the camp doesn't exists or isn't of the type UintArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsUlongArray

```
1. public bool CanGetCampAsUlongArray (string name, out ulong[] val)
```

If the camp doesn't exists or isn't of the type UbyteArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsSbyteArray

```
1. public bool CanGetCampAsSbyteArray (string name, out sbyte[] val)
```

If the camp doesn't exists or isn't of the type SbyteArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsShortArray

```
1. public bool CanGetCampAsShortArray (string name, out short[] val)
```

If the camp doesn't exists or isn't of the type ShortArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsIntArray

```
1. public bool CanGetCampAsIntArray (string name, out int[] val)
```

If the camp doesn't exists or isn't of the type IntArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsLongArray

```
1. public bool CanGetCampAsLongArray (string name, out long[] val)
```

If the camp doesn't exists or isn't of the type LongArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsFloatArray

```
1. public bool CanGetCampAsFloatArray (string name, out float[] val)
```

If the camp doesn't exists or isn't of the type FloatArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsDoubleArray

```
1. public bool CanGetCampAsDoubleArray (string name, out double[] val)
```

If the camp doesn't exists or isn't of the type DoubleArray, it will return false. If it is possible to get the value, it will return true and output it in the out argument

CanGetCampAsDate

```
1. public bool CanGetCampAsDate (string name, out Date val)
```

If the camp doesn't exists or isn't of the type Date, it will return false. If it is possible to get the value, it will return true and output it in the out argument

GetCampType

```
1. public Type GetCampType (string name)
```

Returns the Type of value of a camp. If the camp doesn't exists, an Invalid Type will be returned

CanGetCampType

```
1. public bool CanGetCampType (string name, out Type t)
```

If the camp doesn't exists, returns false. If it is possible to get the type of value, it will return true and output the type in the out argument

DeleteCamp

```
1. public void DeleteCamp (string name)
```

Deletes the camp with that name if it exists

CanDeleteCamp

```
1. public bool CanDeleteCamp (string name)
```

Deletes the camp with that name if it exists and output true, else will output false

RenameCamp

```
1. public void RenameCamp (string oldName, string newName)
```

Renames the camp with the new name if it exists

CanRenameCamp

```
1. public bool CanRenameCamp (string oldName, string newName)
```

Renames the camp with the new name if it exists and output true, else will output false

Equals

```
1. public override bool Equals (object obj)
```

Checks if the contents of two AshFiles are the same

Operators

Operator == (AshFile, AshFile)

```
1. public static bool operator == (AshFile a1, AshFile a2)
```

Checks if the contents of two AshFiles are the same

Operator != (AshFile, AshFile)

```
1. public static bool operator != (AshFile a1, AshFile a2)
```

Checks if the contents of two AshFiles are not the same

Operator + (AshFile, AshFile)

```
1. public static AshFile operator + (AshFile a1, AshFile a2)
```

Will merge the two data structures. If both contain a camp with the same name, the second operand (a2) will have priority and put its camps over

Explicit Operator Dictionary<string, CampValue> (AshFile)

```
1. public static explicit operator Dictionary<string, CampValue> (AshFile af)
```

Casts to a dictionary. Will just return the AshFile's data

Implicit Operator AshFile (Dictionary<string, CampValue>)

```
1. public static implicit operator AshFile (Dictionary<string, CampValue> d)
```

Casts a dictionary into an AshFile. Create a new AshFile with the Dictionary as data