
ByteSize

`ByteSize` is a utility class that makes byte size representation in code easier by removing ambiguity of the value being represented.

`ByteSize` is to bytes what `System.TimeSpan` is to time.

nuget v2.1.2

Development

- Install .NET Core SDK
- Build: `dotnet build`
- Test: `dotnet test`

v2 Breaking Changes

Ratio Changes (HUGE BREAKING CHANGE)

By default `ByteSize` now assumes `1 KB == 1000 B` and `1 KiB == 1024 B` to adhere to the IEC and NIST standards (https://en.wikipedia.org/wiki/Binary_prefix). In version 1 `ByteSize` assumed `1 KB == 1024 B`, that means if you're upgrading from v1, you'll see differences in values.

When you upgrade an existing application to v2 your existing code will be using the decimal representation of bytes (i.e. `1 KB == 1000 B`). If the difference in calculation is not material to your application, you don't need to change anything.

However, if you want to use `1 KiB == 1024 B`, then you'll need to change all `ByteSize` calls to the respective method. For example, calls to `ByteSize.FromKiloByte` need to be changed to `ByteSize.FromKibiByte`.

Lastly, `ByteSize` no longer supports the ratio of `1 KB == 1024 B`. Note this is *kilo*_bytes_ to bytes. The only ratio of `1 == 1024` is *kibi*_bytes_ to bytes.

Other Breaking Changes

- Renamed property `LargestWholeNumberSymbol` and `LargestWholeNumberValue` to `LargestWholeNumberDecimalSymbol` and `LargestWholeNumberDecimalValue` respectively.
- Drop support for all platforms except `netstandard1.0` and `net45`.

Usage

`ByteSize` adheres to the IEC standard, see this [Wikipedia article](#). That means `ByteSize` assumes:

- Decimal representation: 1 `kilobyte` = 1000 `bytes` with 2 letter abbreviations `b`, `B`, `KB`, `MB`, `GB`, `TB`, `PB`.
- Binary representation: 1 `kibibyte` = 1024 `bytes` with 3 letter abbreviations `b`, `B`, `KiB`, `MiB`, `GiB`, `TiB`, `PiB`.

`ByteSize` manages conversion of the values internally and provides methods to create and retrieve the values as needed. See the examples below.

Example

Without `ByteSize`:

```
1 double maxFileSizeMBs = 1.5;
2
3 // I need it in KBs and KiBs!
4 var kilobytes = maxFileSizeMBs * 1000; // 1500
5 var kibibytes = maxFileSizeMBs * 1024; // 1536
```

With `ByteSize`:

```
1 var maxFileSize = ByteSize.FromMegaBytes(1.5);
2
3 // I have it in KBs and KiBs!!
4 maxFileSize.KiloBytes; // 1500
5 maxFileSize.KibiBytes; // 1464.84376
```

`ByteSize` behaves like any other struct backed by a numerical value allowing arithmetic operations between two objects.

```
1 // Add
2 var monthlyUsage = ByteSize.FromGigaBytes(10);
3 var currentUsage = ByteSize.FromMegaBytes(512);
4 ByteSize total = monthlyUsage + currentUsage;
5
6 total.Add(ByteSize.FromKiloBytes(10));
7 total.AddGigaBytes(10);
8
9 // Subtract
10 var delta = total.Subtract(ByteSize.FromKiloBytes(10));
11 delta = delta - ByteSize.FromGigaBytes(100);
12 delta = delta.AddMegaBytes(-100);
```

```
13
14 // Multiple
15 var multiple = ByteSize.FromBytes(4) * ByteSize.FromBytes(2); // 8
16
17 // Divide
18 var divide = ByteSize.FromBytes(16) / ByteSize.FromBytes(8); // 2
```

Constructors

You can create a `ByteSize` object from `bits`, `bytes`, `kilobytes`, `megabytes`, `gigabytes`, and `terabytes`.

```
1 new ByteSize(15);           // Constructor takes in bits (long)
2 new ByteSize(1.5);          // ... or bytes (double)
3
4 // Static Constructors
5 ByteSize.FromBits(10);      // Same as constructor
6 ByteSize.FromBytes(1.5);    // Same as constructor
7
8 // Decimal: 1 KB = 1000 B
9 ByteSize.FromKiloBytes(1.5);
10 ByteSize.FromMegaBytes(1.5);
11 ByteSize.FromGigaBytes(1.5);
12 ByteSize.FromTeraBytes(1.5);
13
14 // Binary: 1 KiB = 1024 B
15 ByteSize.FromKibiBytes(1.5);
16 ByteSize.FromMebiBytes(1.5);
17 ByteSize.FromGibiBytes(1.5);
18 ByteSize.FromTebiBytes(1.5);
```

Properties

A `ByteSize` object contains representations in:

- `bits`, `bytes`
- `kilobytes`, `megabytes`, `gigabytes`, and `terabytes`
- `kibibytes`, `mebibytes`, `gibibytes`, and `tebibytes`

```
1 var maxFileSize = ByteSize.FromKiloBytes(10);
2
3 maxFileSize.Bits;           // 80000
4 maxFileSize.Bytes;         // 10000
5
6 // Decimal
7 maxFileSize.KiloBytes;     // 10
```

```
8 maxSize.MegaBytes; // 0.01
9 maxSize.GigaBytes; // 1E-05
10 maxSize.TeraBytes; // 1E-08
11
12 // Binary
13 maxSize.KibiBytes; // 9.765625
14 maxSize.MebiBytes; // 0.0095367431640625
15 maxSize.GibiBytes; // 9.31322574615479E-06
16 maxSize.TebiBytes; // 9.09494701772928E-09
```

A `ByteSize` object also contains four properties that represent the largest whole number symbol and value.

```
1 var maxSize = ByteSize.FromKiloBytes(10);
2
3 maxSize.LargestWholeNumberDecimalSymbol; // "KB"
4 maxSize.LargestWholeNumberDecimalValue; // 10
5 maxSize.LargestWholeNumberBinarySymbol; // "KiB"
6 maxSize.LargestWholeNumberBinaryValue; // 9.765625
```

String Representation

By default a `ByteSize` object uses the decimal value for string representation.

All string operations are localized to use the number decimal separator of the culture set in `Thread.CurrentCulture`.

ToString `ByteSize` comes with a handy `Tostring` method that uses the largest metric prefix whose value is greater than or equal to 1.

```
1 // By default the decimal values are used
2 ByteSize.FromBits(7).ToString(); // 7 b
3 ByteSize.FromBits(8).ToString(); // 1 B
4 ByteSize.FromKiloBytes(.5).ToString(); // 500 B
5 ByteSize.FromKiloBytes(999).ToString(); // 999 KB
6 ByteSize.FromKiloBytes(1000).ToString(); // 1 MB
7 ByteSize.FromGigabytes(.5).ToString(); // 500 MB
8 ByteSize.FromGigabytes(1000).ToString(); // 1 TB
9
10 // Binary
11 ByteSize.Parse("1.55 kb").ToString("kib"); // 1.51 kib
```

Formatting The `Tostring` method accepts a single `string` parameter to format the output. The formatter can contain the symbol of the value to display.

-
- Base: `b`, `B`
 - Decimal: `KB`, `MB`, `GB`, `TB`
 - Binary: `KiB`, `MiB`, `GiB`, `TiB`

The formatter uses the built in `double.ToString` method.

The default number format is `0.##` which rounds the number to two decimal places and outputs only `0` if the value is `0`.

You can include symbol and number formats.

```
1 var b = ByteSize.FromKiloBytes(10.505);
2
3 // Default number format is 0.##
4 b.ToString("KB");           // 10.52 KB
5 b.ToString("MB");           // .01 MB
6 b.ToString("b");            // 86057 b
7
8 // Default symbol is the largest metric prefix value >= 1
9 b.ToString("#.##");         // 10.5 KB
10
11 // All valid values of double.ToString(string format) are acceptable
12 b.ToString("0.0000");       // 10.5050 KB
13 b.ToString("000.00");       // 010.51 KB
14
15 // You can include number format and symbols
16 b.ToString("#.#### MB");    // .0103 MB
17 b.ToString("0.00 GB");      // 0 GB
18 b.ToString("#.## B");       // 10757.12 B
19
20 // ByteSize object of value 0
21 var zeroBytes = ByteSize.FromKiloBytes(0);
22 zeroBytes.ToString();        // 0 b
23 zeroBytes.ToString("0 kb");  // 0 kb
24 zeroBytes.ToString("0.## mb"); // 0 mb
```

Parsing `ByteSize` has a `Parse` and `TryParse` method similar to other base classes.

Like other `TryParse` methods, `ByteSize.TryParse` returns `boolean` value indicating whether or not the parsing was successful. If the value is parsed it is output to the `out` parameter supplied.

```
1 ByteSize output;
2 ByteSize.TryParse("1.5mb", out output);
3 ByteSize.TryParse("1.5mib", out output);
4
5 // Invalid
6 ByteSize.Parse("1.5 b"); // Can't have partial bits
7
8 // Valid
```

```
9  ByteSize.Parse("5b");
10 ByteSize.Parse("1.55B");
11 ByteSize.Parse("1.55KB");
12 ByteSize.Parse("1.55 kB "); // Spaces are trimmed
13 ByteSize.Parse("1.55 kb");
14 ByteSize.Parse("1.55 MB");
15 ByteSize.Parse("1.55 mB");
16 ByteSize.Parse("1.55 mb");
17 ByteSize.Parse("1.55 GB");
18 ByteSize.Parse("1.55 gB");
19 ByteSize.Parse("1.55 giB");
20 ByteSize.Parse("1.55 TiB");
21 ByteSize.Parse("1.55 tiB");
22 ByteSize.Parse("1.55 tib");
23 ByteSize.Parse("1,55 kib"); // de-DE culture
```

Author and License Omar Khudeira (<http://omar.io>)

Copyright (c) 2013-2022 Omar Khudeira. All rights reserved.

Released under MIT License (see LICENSE file).