Converts a numeric value into number expressed as a size value in bytes, kilobytes, megabytes, gigabytes, or terabytes depending on the size.

```
#region License
// Copyright (c) 2007 James Newton-King
//
// Permission is hereby granted, free of charge, to any person
// obtaining a copy of this software and associated documentation
// files (the “Software”), to deal in the Software without
// restriction, including without limitation the rights to use,
// copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the
// Software is furnished to do so, subject to the following
// conditions:
//
// The above copyright notice and this permission notice shall be
// included in all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND,
// EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
```

```
// OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
// WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
// FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
// OTHER DEALINGS IN THE SOFTWARE.
#endregion

using System;
using System.Globalization;

namespace Newtonsoft.Utilities.Text
{
public class FormatUtils
{
///

/// Converts a numeric value into a string that represents the number
/// expressed as a size value in bytes, kilobytes, megabytes, gigabytes,
/// or terabytes depending on the size. Output is identical to
/// StrFormatByteSize() in shlwapi.dll. This is a format similar to
/// the Windows Explorer file Properties page. For example:
/// 532 becomes 532 bytes
/// 1240 becomes 1.21 KB
/// 235606 becomes 230 KB
/// 5400016 becomes 5.14 MB
///
///
/// It was surprisingly difficult to emulate the StrFormatByteSize() function
/// due to a few quirks. First, the function only displays three digits:
/// – displays 2 decimal places for values under 10 (e.g. 2.12
```

```
KB)
/// – displays 1 decimal place for values under 100 (e.g. 88.2
KB)
/// – displays 0 decimal places for values under 1000 (e.g.
532 KB)
/// – jumps to the next unit of measure for values over 1000
(e.g. 0.97 MB)
/// The second quirk: insiginificant digits are truncated
rather than
/// rounded. The original function likely uses integer math.
/// This implementation was tested to 100 TB.
///
public static string FileSizeToString(long fileSize)
{
if (fileSize < 1024) { return string.Format("{0} bytes",
fileSize); } else { double value = fileSize; value = value /
1024; string unit = "KB"; if (value >= 1000)
{
value = Math.Floor(value);
value = value / 1024;
unit = “MB”;
}
if (value >= 1000)
{
value = Math.Floor(value);
value = value / 1024;
unit = “GB”;
}
if (value >= 1000)
{
value = Math.Floor(value);
value = value / 1024;
unit = “TB”;
}

if (value < 10) { value = Math.Floor(value * 100) / 100;
return string.Format("{0:n2} {1}", value, unit); } else if
```

(value < 100) { value = Math.Floor(value * 10) / 10; return string.Format("{0:n1} {1}", value, unit); } else { value = Math.Floor(value * 1) / 1; return string.Format("{0:n0} {1}", value, unit); } } } } } [/csharp]