Energy Core Library

Filip Golewski

Oct 06, 2021

Contents

CHAPTER 1

Introduction

Energy.Core is a **.NET** class library which boosts your program with functionality covering various type conversions, utility classes, database abstraction layer, object mapping and simple application framework.

To be used by wide range of applications, services, web or console programs.

Filled with radioactive rays

Designed for all purposes as a set of different classes compiled into one library file.

It has minimal set of external dependencies and is available for several .NET platform versions.

Supports multithreading and asynchronous design patterns.

1.1 Installation

The easiest way is install using **nuget** either by finding **Energy.Core** package in official gallery or by executing command.

```
Install-Package Energy.Core
```

Installation package contains versions for **.NET 4**, **.NET Standard / .NET Core** and legacy **.NET 2**. Nuget should choose apropriate version automatically.

1.2 Documentation

http://energy-core.readthedocs.io/

1.3 Examples

1.3.1 Safely converting value types

Conversion functions are located in **Energy.Base.Cast** class. These functions will try to convert value to desired type or use defaults if value cannot be converted.

```
int numberInt = Energy.Base.Cast.StringToInteger("123");
long numberLong = Energy.Base.Cast.StringToLong("1234567890");
double numberDouble = Energy.Base.Cast.StringToDouble("3.1415"); // or "3,1415"
Console.WriteLine(Energy.Base.Cast.DoubleToString(numberDouble));
```

Last will always result in culture invariant version “3.1415”.

1.3.2 Displaying bytes in pretty format

```
byte[] array = Energy.Base.Random.GetRandomByteArray(40);
Console.WriteLine(Energy.Base.Hex.Print(array));
```

Which may result in example output.

```
2c c5 31 be  de 96 fb 5a  76 53 b7 84  2c 09 8d 16   ,.1....ZvS..,...
88 0f c5 6c  50 c3 69 51  48 99 4b 9f  53 00 79 89   ...lP.iQH.K.S y.
1d c9 de c6  4a c9 dc e2                            ....J...
```

This was very basic usage but you may extend it with different formatting options, offsets and even coloring.

1.3.3 Waiting for user input on console

When you call **Console.ReadLine()** program will stop and wait for user input. If you want to stop only when user enters data, use **Energy.Core.Tilde.ReadLine()** which will result in *null* as long as user has not accepted its input by pressing Enter key allowing your program to do its job.

1.3.4 Easily make REST requests

How about:

```
string url = "https://www.google.com/search?q=Energy";
Console.WriteLine(Energy.Core.Web.Get(url).Body);
```

Easy to use, build upon standard **System.Net.WebRequest** class REST functions available for common methods like GET, POST, PUT, PATCH, DELETE, HEAD or OPTIONS.

1.3.5 Generic SQL database connection

Here database connection is made using general connection class cooperating with each **ADO.NET** provider of the database connection.

```
Energy.Source.Connection<MySql.Data.MySqlClient.MySqlConnection> db
    = new Energy.Source.Connection<MySql.Data.MySqlClient.MySqlConnection>();
db.ConnectionString = @"Server=127.0.0.1;Database=test;Uid=test;Pwd=test;";
if (!db.Test())
{
    Console.WriteLine("Connection test error");
}
else
{
    string result = db.Scalar<string>("SELECT CURRENT_TIMESTAMP()");
    Console.WriteLine("Current server time is: {0}", result);
}
```

Connections are thread safe, may be cloned or even set to be persistent if you want to limit connections to your SQL database.

1.4 Content

Library has been divided into several different namespaces. Following table briefly describes the purpose of each of them.

- *Energy.Base* - Contains base classes
- *Energy.Core* - Library functions
- *Energy.Attribute* - Attributes
- *Energy.Enumeration* - Enumerations
- *Energy.Interface* - Interfaces
- *Energy.Source* - Database connection

1.5 History

Working for many years on different development projects, from simple applications, web applicatons, to a rich and monolithic production environment with plenty of small software programs that act as interfaces and all kinds of small services, as most of you have probably noticed that some part of the functionality is repeated to a greater or lesser extent regardless of the project type.

This library was created completely independently from my professional work as an attempt to build a “base”, which can be quickly used in almost any project in order not to repeat again the same codes to achieve functionality like safe (international) type conversion or generic database connection which is easy and most importantly safe to use.

Made with love for you

1.6 Greetings

To be continued...

CHAPTER 2

Standard

2.1 Energy Base Standard

2.1.1 Text

2.1.1.1 Trim

Remove all leading and trailing whitespace characters from text.

Following characters are treated as whitespace: space character " " (code 32), horizontal tab "\t" (code 09), line feed "\n" (code 10), carriage return "\r" (code 13), vertical tab "\v" (code 11), form feed "\f" (code 12), null character "\0" (code 0).

2.1.1.2 IsInteger

Check if value represents integer number.

2.1.1.3 IsNumber

Check if value represents number.

12.e3

12313,123E+123

2.1.2 Cast

2.1.2.1 StringToBool

Convert text to boolean value.

2.1.2.2 StringToInt

Convert text to signed 32-bit integer number without exception ignoring leading and trailing whitespace characters.

If conversion cannot be performed, default value 0 is returned.

2.1.2.3 StringToLong

Convert text to signed 64-bit long integer number without exception.

If conversion can't be done, 0 will be returned.

2.2 Energy Core Standard

2.3 Energy Source Standard

CHAPTER 3

Type conversion

3.1 As

Generic conversion from one type to another.

```
T Energy.Base.Cast.As<T>(object value)
```

```
object Energy.Base.Cast.As(System.Type type, object value)
```

3.2 ObjectToDouble

Convert object to double value without exception. Treat comma "," the same as dot "." as decimal point when converting from string.

```
double Energy.Base.Cast.ObjectToDouble(object value)
```

3.3 ObjectToDecimal

Convert object to decimal number. Treat comma "," the same as dot "." as decimal point when converting from string.

```
double Energy.Base.Cast.ObjectToDouble(object value)
```

3.4 ObjectToLong

Convert object to long integer number.

```
long Energy.Base.Cast.ObjectToLong(object value)
```

3.5 ObjectToUnsignedLong

Convert object to long integer number.

```
ulong Energy.Base.Cast.ObjectToUnsignedLong(object value)
```

3.6 ObjectToBool

Convert object to boolean value.

```
bool Energy.Base.Cast.ObjectToBool(object value)
```

3.7 ObjectToJsonValue

Convert object to formal JSON value string.

```
string Energy.Base.Cast.ObjectToJsonValue(object value)
```

3.8 JsonValueToObject

Convert JSON value string to an object.

```
object Energy.Base.Cast.JsonValueToObject(string text)
```

3.9 StringToDecimal

Convert string to decimal value without exception. Treat comma "," the same as dot "." as decimal point.

```
decimal Energy.Base.Cast.StringToDecimal(string value, NumberStyles numberStyles)
```

```
decimal Energy.Base.Cast.StringToDecimal(string value)
```

3.10 StringToLongSmart

Convert string to long integer value without exception removing numerical differences.

```
long Energy.Base.Cast.StringToLongSmart(string value)
```

3.11 StringToUnsignedLongSmart

Convert string to unsigned long integer value without exception removing numerical differences.

```
long Energy.Base.Cast.StringToUnsignedLongSmart(string value)
```

3.12 BoolToString

Convert boolean value to its string representation.

```
string Energy.Base.Cast.BoolToString(bool value, Energy.Enumeration.BooleanStyle␣
↪style)
```

3.13 Enum

Convert string to enumeration value.

```
object Energy.Base.Cast.StringToEnum(string value, Type type)
```

3.14 RemoveNumericalDifferences

Remove numerical differences from text representation of number.

Treat comma "," the same as dot "." as decimal point.

Ignore space, underscore and apostrophes between digits.

```
string Energy.Base.Cast.RemoveNumericalDifferences(string value)
```

3.15 MemorySizeToString

Represents memory size as a string containing a numeric value with an attached size unit.

Units used are: "B", "KB", "MB", "GB", "TB", "PB", "EB".

```
string Energy.Base.Cast.MemorySizeToString(long sizeInBytes, int decimalPlaces, bool␣
↪numberCeiling)
```

3.16 IntegerToHex

Convert integer to hexadecimal value.

Resulting string will have count specified by size of digits or letters (A-F).

For **int** which is the synonym of **System.Int32** resulting string will be 8 characters long.

If number representation will be larger than size, it will be truncated to the last characters.

Example: IntegerToHex(100000, 4) will result with “86a0” instead of “186a0” or “186a”.

```
string Energy.Base.Cast.IntegerToHex(int value)
```

3.17 HexToInteger

Convert hexadecimal string to integer value (System.Int32).

```
int Energy.Base.Cast.HexToInteger(string hex)
```

3.18 OctToInteger

Convert octal string to integer value (System.Int32).

```
int Energy.Base.Cast.OctToInteger(string oct)
```

3.19 StringToStream

Convert string to a stream using specified encoding.

```
Stream Energy.Base.Cast.StringToStream(string value, Encoding encoding)
```

If encoding is not specified, UTF-8 will be used.

```
Stream Energy.Base.Cast.StringToStream(string value)
```

Always try to use streams in using section to properly free allocated objects like in the following example of unit test using this function.

```
string needle = "€";
byte[] buffer;

using (Stream stream = Energy.Base.Cast.StringToStream(needle))
{
    int length = (int)stream.Length;
    buffer = new byte[length];
    stream.Read(buffer, 0, length);
}
Assert.AreEqual(0, Energy.Base.Bit.Compare(new byte[] { 226, 130, 172 }, buffer));

using (Stream stream = Energy.Base.Cast.StringToStream(needle, encoding: Encoding.
↪Unicode))
{
    int length = (int)stream.Length;
    buffer = new byte[length];
    stream.Read(buffer, 0, length);
}
Assert.AreEqual(0, Energy.Base.Bit.Compare(new byte[] { 172, 32 }, buffer));
```

3.20 StringToShort

Convert string to short integer value without exception.

```
ushort Energy.Base.Cast.StringToShort(string value)
```

You may want to set additional options like allow decimal point in numbers or exceeding value. When value exceeds maximum, reminder will be returned.

```
ushort Energy.Base.Cast.StringToShort(string value, bool point, bool exceed)
```

3.21 StringToUnsignedShort

Convert string to short integer value without exception.

```
ushort Energy.Base.Cast.StringToUnsignedShort(string value)
```

You may want to set additional options like allow decimal point in numbers or exceeding value. When value exceeds maximum, reminder will be returned. In addition when *exceed* parameter is true, negative values will be returned as positive.

```
ushort Energy.Base.Cast.StringToUnsignedShort(string value, bool point, bool exceed)
```

CHAPTER 4

Text functions

Text related functions.

There was a small question whether the class name should be renamed from **Energy.Base.Text** to something else to avoid possible conflicts with **System.Text** when anyone wants to add **Energy.Base** namespace to *using* list. It was decided to keep it as it is while recommending using full class names in *using* list and synonyms as well.

4.1 Constants

HTML break.

```
Energy.Base.Text.BR = "<br>";
```

New line string. *CR LF* only for Windows, *LF* otherwise.

```
Energy.Base.Text.NL = "\r\n" | "\n"
```

Whitespace characters string.

```
Energy.Base.Text.WS = " \t\r\n\v";
```

An array of empty texts containing end-of-line characters.

```
Energy.Base.Text.NEWLINE_ARRAY = new string[] { "\r\n", "\n", "\r" };
```

Regular expression pattern for new line.

```
Energy.Base.Text.NEWLINE_PATTERN = "\r\n|\n|\r";
```

4.2 Quotation

Introducing specialized class **Energy.Base.Text.Quotation** to help with quotation in texts.

```
"Hello ""John""..."
```

Use static method **Energy.Base.Text.Quotation.From** to create object directly from text identifier.

```
var q = Energy.Base.Text.Quotation.From("<<>>");
```

This method will create quotation definition object from a string. If string is null or empty, null will be returned.

If string contains only 1 character, it would be treated as prefix and suffix character, double suffix character will be used as escape sequence. If definition contains spaces but not starts with, it will be splited by it. First element of such array will be used as prefix, last one as suffix, and all elements between them will be treated as escape sequences. If the number of characters is even, first half will be treated as prefix, second as suffix, double suffix will be treated as escape sequence. If the number of characters is odd, middle character will be treated as escape character for suffix sequence of characters. It's common to use backslash there.

4.3 Cut

Cut or return part of text which ends with one of ending sequences, supporting optional quotations.

If text contains part in quotes, it will be included as is, together with quotation characters until ending sequence is found.

When cutting text "abHello '$'$d" by dollar sign ($) as ending and apostrophes (') as quotations text will be cutted in following pieces: "a", "b", "Hello '$'", "d".

```
string Energy.Base.Text.Cut(string text, string[] terminator, string[] quotation)
```

When passing text as reference, original object will be replaced with what left. Use it in a loop until resulting text is empty to cut all parts.

```
string Energy.Base.Text.Cut(ref string text, string[] terminator, string[] quotation)
```

4.4 Exchange

Exchange texts or characters between each other.

```
void Energy.Base.Text.Exchange(ref string first, ref string second)
```

```
void Energy.Base.Text.Exchange(ref char first, ref char second)
```

4.5 Select

Select first non empty string element.

```
string Energy.Base.Text.Select(params string[] list)
```

4.6 Cell

Align and limit the text to the specified size. Cut the initial characters from the text value. If there are enough space, add a prefix and a suffix in order from the alignment direction of the text.

Some of parameters used are:

- *text* : **string**

  Text value to be aligned in a cell. That's obvious.

- *start* : **int**

  The initial index of the text to be cut out. If less than zero, it indicates the last characters of the text.

- *fill* : **char**

  Character that will be used if text is shorter than specified size.

- *pad* : **Energy.Enumeration.TextPad**

  Padding direction, may be left or right. Because padding is defined as flags, center or middle is also avaiable.

- *prefix* : **string**

  Optional prefix text that can be added if there is a space in resulting text to match size.

- *suffix* : **string**

  Optional suffix text that can be added if there is a space in resulting text to match size.

There is also a version of **Energy.Base.Text.Cell** function which can return remains of text that did not fit in the specified size of text.

```
string Energy.Base.Text.Cell(string text, int start, int size, Energy.Enumeration.
↪TextPad pad, char fill, string prefix, string suffix, out string remains)
```

```
string Energy.Base.Text.Cell(string text, int start, int size, Energy.Enumeration.
↪TextPad pad, out string remains)
```

```
string Energy.Base.Text.Cell(string text, int start, int size, Energy.Enumeration.
↪TextPad pad, char fill, out string remains)
```

```
string Energy.Base.Text.Cell(string text, int start, int size, Energy.Enumeration.
↪TextPad pad)
```

```
string Energy.Base.Text.Cell(string text, int start, int size, Energy.Enumeration.
↪TextPad pad, char fill)
```

```
string Energy.Base.Text.Cell(string text, int size, Energy.Enumeration.TextPad pad,␣
↪out string remains)
```

```
string Energy.Base.Text.Cell(string text, int size, Energy.Enumeration.TextPad pad,␣
↪char fill, out string remains)
```

```
string Energy.Base.Text.Cell(string text, int size, Energy.Enumeration.TextPad pad)
```

```
string Energy.Base.Text.Cell(string text, int size, Energy.Enumeration.TextPad pad,␣
↪char fill)
```

The same with *Energy.Enumeration.TextAlign* instead of *Energy.Enumeration.TextPad*.

```
string Energy.Base.Text.Cell(string text, int start, int size, Energy.Enumeration.
↪TextAlign align, out string remains)
```

```
string Energy.Base.Text.Cell(string text, int start, int size, Energy.Enumeration.
↪TextAlign align)
```

```
string Energy.Base.Text.Cell(string text, int start, int size, Energy.Enumeration.
↪TextAlign align, char fill)
```

```
string Energy.Base.Text.Cell(string text, int size, Energy.Enumeration.TextAlign␣
↪align, out string remains)
```

4.6.1 Example

```
string text;
text = "Ana's Song";
string cell;
cell = Energy.Base.Text.Cell(text, 0, 3, Energy.Enumeration.TextPad.Left);
// cell is now "Ana"
cell = Energy.Base.Text.Cell(text, -4, 4, Energy.Enumeration.TextPad.Left);
// cell is now "Song"
```

4.7 EscapeExpression

Escape text for regular expression.

```
string Energy.Base.Text.EscapeExpression(string text)
```

4.8 Random

Generate random text. Resulting string will contain upper and lower latin letters and numbers only. You may expect length from 3 to 10 characters.

```
string Energy.Base.Text.Random()
```

4.9 Surround

Surround text with prefix and suffix, optionally adding prefix only when needed.

- *text* : **string**

  Text to surround.

- *prefix* : **string**

  Prefix to add at begining.

- *suffix* : **string**

  Suffix to add at ending.

- *optional* : **bool**

  Add prefix and suffix only when needed.

```
string Energy.Base.Text.Surround(string text, string prefix, string suffix, bool␣
↪optional)
```

4.10 Texture

Repeat string pattern to specified amount of characters length.

```
string Energy.Base.Text.Texture(string pattern, int size)
```

4.11 Trim

Remove leading and trailing whitespace. Includes space, tabulation (horizontal and vertical), new line and null characters.

```
string Energy.Base.Text.Trim(string value)
```

4.12 IsWild

Check if string contains one of wild characters ("*" or "?").

```
bool Energy.Base.Text.IsWild(string text)
```

4.13 IsLike

Check if string contains one of characters used in LIKE ("%" or "_").

```
bool Energy.Base.Text.IsLike(string text)
```

4.14 ConvertNewLine

Convert new line delimiter to specified one.

```
string Energy.Base.Text.ConvertNewLine(string text, string newLine)
```

Convert newline delimiter to environment default. Value of constant **Energy.Base.Text.NL** is used.

```
string Energy.Base.Text.ConvertNewLine(string text)
```

4.15 Join

Join strings into one list with separator.

For example Energy.Base.Text.Join(" : ", "A", "B", "", "C") will return "A : B : : C".

```
string Energy.Base.Text.Join(string glue, bool empty, params string[] array)
```

Join non empty and optionally empty strings into one list with separator.

For example Energy.Base.Text.Join(" : ", false, "A", "B", "", "C") will return "A : B : C".

```
string Energy.Base.Text.Join(string glue, params string[] array)
```

Join elements of string dictionary.

```
string Energy.Base.Text.Join(string glue, string format, Dictionary<string, string>␣
↪dictionary)
```

```
Dictionary<string, string> d = new Dictionary<string, string>();
d["a"] = "B";
d["c"] = "D";
string s = Energy.Base.Text.Join(", ", "{0}-{1}", d); // "a-B, c-D"
```

Join elements of string-object dictionary.

```
string Energy.Base.Text.Join(string glue, string format, Dictionary<string, object>␣
↪dictionary)
```

4.16 Quote

Surround text with quotation characters (").

Escape existing quotation characters with additional quotation character ("").

```
string Energy.Base.Text.Quote(string text)
```

Surround text with quotation characters (") optionally.

If optional parameter is true, text will be quoted only when text contains specified quotation character. Escape existing quotation characters with additional quotation character ("").

```
string Energy.Base.Text.Quote(string text, bool optional)
```

Surround text with specified quotation characters optionally.

If optional parameter is true, text will be quoted only when text contains specified quotation character. Escape existing quotation characters with additional quotation character.

```
string Energy.Base.Text.Quote(string text, string with, bool optional)
```

Surround text with specified quotation characters.

Escape existing quotation character with specified escape character.

```
string Energy.Base.Text.Quote(string text, string with, string escape)
```

4.16.1 Example

```
// Excpect 'Hello\'John\'.'
Debug.WriteLine(Energy.Base.Text.Quote("Hello 'John'.", "'", "\\"));
```

4.17 Strip

Strip text from double quotation marks.

Two sequential quotation marks inside the text will be replaced by single characters.

```
string Energy.Base.Text.Strip(string text)
```

Strip text from quotation.

Two sequential quotation characters inside the text will be replaced by single characters.

```
string Energy.Base.Text.Strip(string text, char quote)
```

```
string Energy.Base.Text.Strip(string text, string quote)
```

Strip text from quotation.

Escape character for including quotation characters inside the text may be provided.

```
string Energy.Base.Text.Strip(string text, char quote, char escape)
```

```
string Energy.Base.Text.Strip(string text, string quote, string escape)
```

Strip text from quotation.

Escape character for including quotation characters inside the text may be provided.

```
string Energy.Base.Text.Strip(string text, string start, string end, string escape,␣
↪out bool change)
```

```
string Energy.Base.Text.Strip(string text, string start, string end, string escape)
```

Strip text from quotation.

Escape character for including quotation characters inside the text may be provided.

Multiple variations may be set to allow different quotation styles to work.

```
string Energy.Base.Text.Strip(string text, string[] start, string[] end, string[]␣
↪escape, out bool change)
```

```
string Energy.Base.Text.Strip(string text, string[] start, string[] end, string[]␣
↪escape)
```

Example

```
var qs = new string[] { "'", "`" };
var s1 = Energy.Base.Text.Strip("'It''s a dog eat dog world.'", qs, qs, qs);
var s2 = Energy.Base.Text.Strip("`It``s a dog eat dog world.`", qs, qs, qs);
```

```
var qa = new string[] { "@\"", "\"" };
var qb = new string[] { "\"", "\"" };
var qe = new string[] { "\"\"", "\\" };
var s1 = Energy.Base.Text.Strip("@\"Verbatim \"\"style\"\" example", qa, qb, qe);
var s2 = Energy.Base.Text.Strip("\"Normal \\\"style\\\" example", qa, qb, qe);
```

4.18 HasDigitsOnly

Checks if string contains only digits.

```
bool Energy.Base.Text.HasDigitsOnly(string value)
```

4.19 IsInteger

Checks if string is an integer number.

```
bool Energy.Base.Text.IsInteger(string value, bool negative)
```

```
bool Energy.Base.Text.IsInteger(string value)
```

4.20 IsLong

Checks if string is a long integer number.

```
bool Energy.Base.Text.IsLong(string value, bool negative)
```

```
bool Energy.Base.Text.IsLong(string value)
```

4.21 EmptyIfNull

Return empty string when string parameter is null or string parameter itself otherwise. This function ensures string objects are always defined.

```
string Energy.Base.Text.EmptyIfNull(string value)
```

4.22 IfEmpty

Returns first non empty string from a parameter list. Strings are considered to be empty if they are null. Function will return empty string ("") if parameter list is empty.

```
string Energy.Base.Text.TextIfEmpty(params string[] input)
```

4.23 IfWhite

Returns first non white string from a parameter list. White string is considered to be null, zero length or string containing only whitespace characters. Function will return empty string ("") if parameter list is empty.

```
string Energy.Base.Text.TextIfWhite(params string[] input)
```

4.24 RemoveWhite

Remove whitespace characters from entire string.

```
string Energy.Base.Text.RemoveWhite(string value)
```

4.25 ContainsWhite

Check if text or character array contains whitespace.

```
string Energy.Base.Text.ContainsWhite(char[] array)
```

```
string Energy.Base.Text.ContainsWhite(string text)
```

4.26 ReplaceWhite

Replace whitespace characters with replacement string in entire string.

```
string Energy.Base.Text.ReplaceWhite(string text, string replacement)
```

4.27 Encoding

Find System.Text.Encoding for specified name.

Get System.Text.Encoding.UTF8 by default or if encoding not exists.

Treats UCS-2 the same as UTF-16 besides differences.

Accepts values like "UTF-8", "UTF", "UTF8", "UNICODE", "UCS-2 LE", "UCS-2 BE", "1250", "1252", and so on.

```
System.Text.Encoding Energy.Base.Text.Encoding(string encoding)
```

4.28 Capitalize

Return a word with its first letter upper case and remaining letters in lower case.

```
string Energy.Base.Text.Capitalize(string word)
```

Return array of words with their first letters upper case and remaining letters in lower case.

```
string Energy.Base.Text.Capitalize(string word)
```

4.29 Upper

Change letters in a word to upper case.

```
string Energy.Base.Text.Upper(string word)
```

Change letters in word list to upper case.

```
string Energy.Base.Text.Upper(string[] words)
```

The reason why this function is included is that it works with unicode.

4.30 Lower

Change letters in a word to lower case.

```
string Energy.Base.Text.Lower(string[] words)
```

Change letters in word list to lower case.

```
string Energy.Base.Text.Lower(string[] words)
```

The reason why this function is included is that it works with unicode also.

4.31 IncludeLeading

Include leading text if not already present at the begining.

```
string Energy.Base.Text.IncludeLeading(string text, string missing)
```

```
string Energy.Base.Text.IncludeLeading(string text, string missing, bool ignoreCase)
```

Include leading character if not already present at the begining.

```
string Energy.Base.Text.IncludeLeading(string text, char missing)
```

```
string Energy.Base.Text.IncludeLeading(string text, char missing, bool ignoreCase)
```

4.32 IncludeTrailing

Include trailing text if not already present at the end.

```
string Energy.Base.Text.IncludeTrailing(string text, string missing)
```

```
string Energy.Base.Text.IncludeTrailing(string text, string missing, bool ignoreCase)
```

Include leading character if not already present at the end.

```
string Energy.Base.Text.IncludeTrailing(string text, char missing)
```

```
string Energy.Base.Text.IncludeTrailing(string text, char missing, bool ignoreCase)
```

4.33 RemoveEmptyLines

Remove empty lines from string.

```
string Energy.Base.Text.RemoveEmptyLines(string text)
```

4.34 RemoveEmptyElements

Remove empty elements from array of strings.

New array will be returned.

```
string[] Energy.Base.Text.RemoveEmptyElements(string[] array)
```

Remove empty elements from list of strings.

List will be modified and returned back.

```
void Energy.Base.Text.RemoveEmptyElements(List<string> list)
```

4.35 Contains

Check if text contains searched string.

```
bool Energy.Base.Text.Contains(string text, string search, bool ignoreCase)
```

```
bool Energy.Base.Text.Contains(string text, string search)
```

Check if text representation of object contains searched string.

```
bool Energy.Base.Text.Contains(object o, string search, bool ignoreCase)
```

```
bool Energy.Base.Text.Contains(object o, string search)
```

CHAPTER 5

Classes, objects and assemblies

Energy.Base.Class contains functions related to classes, objects and assemblies.

You will find here functions to modify fields or properties of C# objects.

5.1 GetDefault

Get default value of specified type.

```
object Energy.Base.Class.GetDefault(System.Type type)
```

```
T Energy.Base.Class.GetDefault<T>()
```

5.2 GetFieldsAndProperties

Get list of names of all fields and properties of specified class type.

```
string[] Energy.Base.Class.GetFieldsAndProperties(Type type, bool includePrivate,␣
↪bool includePublic)
```

```
string[] Energy.Base.Class.GetFieldsAndProperties(Type type, bool includePrivate)
```

Get list of names of public fields and properties of specified class type.

```
string[] Energy.Base.Class.GetFieldsAndProperties(Type type)
```

5.3 GetFieldOrPropertyAttribute

Get first attribute value for a field or property of desired class.

```
object Energy.Base.Class.GetFieldOrPropertyAttribute(Type type, string name, Type␣
↪attribute)
```

```
T Energy.Base.Class.GetFieldOrPropertyAttribute<T>(Type type, string name)
```

5.4 GetFieldOrPropertyAttributes

Get custom attributes of field or property of a class.

```
object[] Energy.Base.Class.GetFieldOrPropertyAttributes(Type type, string field, Type␣
↪filter, bool inherit, bool ignoreCase)
```

```
T[] Energy.Base.Class.GetFieldOrPropertyAttributes<T>(Type type, string field, bool␣
↪inherit, bool ignoreCase)
```

```
object[] Energy.Base.Class.GetFieldOrPropertyAttributes(Type type, string field, Type␣
↪filter)
```

5.5 GetTypesWithInterface

Filter types that implements specified interface.

```
System.Type[] Energy.Base.Class.GetTypesWithInterface(System.Type[] types, System.
↪Type interfaceType)
```

5.6 GetTypeWithInterface

Get first type that implements specified interface. Return null if not found.

```
System.Type Energy.Base.Class.GetTypeWithInterface(System.Type[] types, System.Type␣
↪interfaceType)
```

5.7 GetTypesWithAttribute

Filter types that have specified attribute.

```
System.Type[] Energy.Base.Class.GetTypesWithAttribute(System.Type[] types, System.
↪Type attributeType)
```

5.8 GetTypeWithAttribute

Get first type having specified attribute. Return null if not found.

```
System.Type Energy.Base.Class.GetTypeWithAttribute(System.Type[] types, System.Type␣
↪attributeType)
```

5.9 GetClassAttribute

Get desired attribute for a class or null if not found.

```
object Energy.Base.Class.GetClassAttribute(Type type, Type attribute)
```

```
T Energy.Base.Class.GetClassAttribute<T>(Type type)
```

5.10 IsStatic

True if class is static and cannot be instantiated.

```
bool Energy.Base.Class.IsStatic(Type type)
```

5.11 IsInstance

True if object of specified class can be created. At least one public constructor needs to be found. Function will result false for for static class type.

```
bool Energy.Base.Class.IsInstance(Type type)
```

5.12 HasParameterlessConstructor

True if class has public parameterless constructor.

```
bool Energy.Base.Class.HasParameterlessConstructor(Type type)
```

5.13 HasParameteredConstructor

True if class has at least one public constructor with specified number parameters.

```
bool Energy.Base.Class.HasParameteredConstructor(Type type, int minimumParameterCount,
↪ int maximumParameterCount)
```

True if class has constructor with one or more parameters.

```
bool Energy.Base.Class.HasParameteredConstructor(Type type)
```

5.14 Mangle

Mangle object by applying a function to each field and property of specified type. Returns number of fields and properties affected.

```
int Energy.Base.Class.Mangle<T>(object subject, bool includePrivate, bool␣
↪includePublic
    , Energy.Base.Anonymous.Function<T, T> function)
```

```
var o = new { Text = " Text " };
Energy.Base.Class.Mangle<string>(o, delegate (string _) { return (_ as string ?? "").
↪Trim(); });
```

Mangle object by applying a function to each public field and property of specified type. Returns number of fields and properties affected.

```
int Energy.Base.Class.Mangle<T>(object subject, Energy.Base.Anonymous.Function<T, T>␣
↪function)
```

Mangle object by applying a function to public field or property of specified class type. Returns 1 if value was changed, 0 if not found or read only.

```
int Energy.Base.Class.Mangle<T>(object subject, string name, bool includePrivate,␣
↪bool includePublic
    , Energy.Base.Anonymous.Function<T, T> function)
```

```
var msg = new TextClass() { Text = "" };
if (1 == Energy.Base.Class.Mangle<string>(msg, "Text", delegate { return "Hello"; }))
{
    Debug.WriteLine("I just set Text field or property to " + msg.Text);
}
var anon = new { Text = (string)null };
if (0 == Energy.Base.Class.Mangle<string>(anon, "Text", delegate { return "Hello"; }))
{
    Debug.WriteLine("I can't change anything in anonymous object");
}
```

Mangle object by applying a function to public field or property of specified class type. Returns 1 if value was changed, 0 if not found or read only.

```
int Energy.Base.Class.Mangle<T>(object subject, string name, Energy.Base.Anonymous.
↪Function<T, T> function)
```

5.15 GetAssemblyFile

Get filename of assembly.

This is just a simple alias for Location field of assembly object.

```
string Energy.Base.Class.GetAssemblyFile(System.Reflection.Assembly assembly)
```

5.16 GetAssemblyDirectory

Get directory name of assembly.

```
string Energy.Base.Class.GetAssemblyDirectory(System.Reflection.Assembly assembly)
```

CHAPTER 6

Command line arguments

Brand new class **Energy.Base.Command.Arguments** appeared here to ease implement command line options for everybody.

This class is build with Named Parameter Idiom design pattern.

Argument parsing mechanism was inspired by excelent npm package **yargs** for JavaScript.

```
var argv = new Energy.Base.Command.Arguments(args)
    .Switch("help")
    .Switch("quiet")
    .Parameter("input")
    .Parameter("output")
    .Alias("?", "help")
    .Alias("q", "quiet")
    .Alias("i", "input")
    .Alias("o", "input")
    .Help("input", "Input file")
    .Help("output", "Output file")
    .Parse();

if (!argv["help"].Empty)
{
    Console.WriteLine("Help");
}
```

6.1 Constructor

```
Energy.Base.Command.Arguments(string[] args)
```

```
Energy.Base.Command.Arguments(string line)
```

You can still add argument line later.

```
Energy.Base.Command.Arguments()
```

6.2 Parameter

Add command line parameter that will consume one or more trailing arguments and provide them as values.

If you set count to 0, it will become switch (flag).

```
Arguments Parameter(string name, int count)
```

Add command line parameter that will consume one next trailing argument and provide it as value.

```
Arguments Parameter(string name)
```

Special one.

Treat all unknown options as parametered.

```
Arguments Parameter()
```

6.3 Switch

Add single command line switch option, known also as flag. Like “-v”, or “–version”.

```
Arguments Switch(string name)
```

Special one.

Treat all unknown options as simple switches.

```
Arguments Switch()
```

6.4 Line

Add arguments from text line.

Arguments are divided by any whitespace character.

Arguments may use double quote (”) character to include whitespace, and multiple quoting is allowed within one argument.

For example: C:”Documents and settings””Program Files”will be considered as one argument.

```
Arguments Line(string line)
```

6.5 Skip

Skip first n entries when parsing arguments.

```
Skip(int skip)
```

6.6 Strict

Set strict mode.

When strict mode is set, exception will be thrown on unrecognized option name.

```
Arguments Strict(bool strict)
```

6.7 Slash

Allow usage of slash options (starting with "/").

Allows to use DOS style options like "/?".

It's RSX-11 (and other similar DEC systems), through CP/M to MS-DOS legacy.

Should be avoided probably.

```
Arguments Slash(bool slash)
```

6.8 Short

Allow usage of short options (starting with "-").

Turned on as default.

```
Arguments Short(bool enable)
```

6.9 Long

Allow usage of long options (starting with "–").

Turned on as default.

```
Arguments Long(bool enable)
```

6.10 Alias

Add alias to parameter key.

```
Arguments Alias(string alias, string name)
```

6.11 Help

Add help description for parameter key.

```
Arguments Help(string name, string description)
```

6.12 Parse

Probably most important function here.

Parse arguments and set values for options.

```
Arguments Parse()
```

Parameterless method will parse arguments set by constructor or modified later.

When calling with parameters, arguments to be parsed will only be taken from invoker.

```
Arguments Parse(string line)
```

```
Arguments Parse(string[] args)
```

CHAPTER 7

Number functions

Mathematical functions.

7.1 Median

Return the middle number from an array.

```
int Energy.Base.Number.Median(int[] array)
```

```
int Energy.Base.Number.Median(int[] array, bool sort)
```

```
long Energy.Base.Number.Median(long[] array)
```

```
long Energy.Base.Number.Median(long[] array, bool sort)
```

```
ulong Energy.Base.Number.Median(ulong[] array)
```

```
ulong Energy.Base.Number.Median(ulong[] array, bool sort)
```

```
double Energy.Base.Number.Median(double[] array)
```

```
double Energy.Base.Number.Median(double[] array, bool sort)
```

```
decimal Energy.Base.Number.Median(decimal[] array)
```

```
decimal Energy.Base.Number.Median(decimal[] array, bool sort)
```

CHAPTER 8

Hashing functions

Several hashing functions for sequences of characters or bytes are available.

8.1 MD5

Return MD5 hash for a string.

```
string Energy.Base.Hash.MD5(string text, Encoding encoding)
```

```
string Energy.Base.Hash.MD5(string text)
```

8.2 SHA-1

Return SHA-1 hash for a string.

```
string Energy.Base.Hash.SHA1(string text, Encoding encoding)
```

```
string Energy.Base.Hash.SHA1(string text)
```

8.3 SHA-256

Return SHA-256 (SHA-2) hash for a string.

```
string Energy.Base.Hash.SHA256(string text, Encoding encoding)
```

```
string Energy.Base.Hash.SHA256(string text)
```

8.4 SHA-384

Return SHA-384 (SHA-2) hash for a string.

```
string Energy.Base.Hash.SHA384(string text, Encoding encoding)
```

```
string Energy.Base.Hash.SHA384(string text)
```

8.5 SHA-512

Return SHA-512 (SHA-2) hash for a string.

```
string Energy.Base.Hash.SHA512(string text, Encoding encoding)
```

```
string Energy.Base.Hash.SHA512(string text)
```

8.6 CRC

For each characters do a 5-bit left circular shift and XOR in character numeric value (CRC variant).

```
string Energy.Base.Hash.CRC(string value)
```

```
string Energy.Base.Hash.CRC2(string value)
```

8.7 CRC-16-CCITT

Calculate 16-bit CRC-CCITT checksum with specified polynominal and initial value.

```
ushort Energy.Base.Hash.CRC16CCITT(byte[] array, ushort poly, ushort init)
```

```
ushort Energy.Base.Hash.CRC16CCITT(string text, ushort poly, ushort init)
```

8.8 PJW

For each characters add character numeric value and left shift by 4 bits (PJW hash).

```
uint Energy.Base.Hash.PJW(string value)
```

CHAPTER 9

Clock, date and time

Energy.Base.Clock contains functions related to classes, objects and assemblies.

You will find here functions to modify fields or properties of C# objects.

9.1 CurrentTime

Return current date in ISO format.

Example: “2020-01-01”.

```
string Energy.Base.Clock.CurrentDate
```

9.2 CurrentTime

Return current time as time string with millisecond part in 24h format.

Example: “17:33:15.176”.

```
string Energy.Base.Clock.CurrentDate
```

9.3 CurrentTimeSpace

Return current time as time string with millisecond part in 24h format and trailing space. Example: “17:33:15.176 “.

```
string Energy.Base.Clock.CurrentTimeSpace
```

9.4 Floor

Round time to specified precision.

```
DateTime Energy.Base.Clock.Floor(DateTime value, int precision)
```

Precision may be positive or negative number. On positive precision function will include as many fractional second digits as possible (up to 7). On negative precision function may round to minute, hour or even a year. Use 3 for milliseconds, 6 for microseconds, -2 for minutes, -4 for hours, etc.

```
DateTime needle;
DateTime result;
needle = new DateTime(2121, 12, 16, 21, 17, 33, 456);
result = Energy.Base.Clock.Floor(needle, 4);   // 2121-12-16 21:17:33.456
result = Energy.Base.Clock.Floor(needle, 3);   // 2121-12-16 21:17:33.456
result = Energy.Base.Clock.Floor(needle, 2);   // 2121-12-16 21:17:33.450
result = Energy.Base.Clock.Floor(needle, 1);   // 2121-12-16 21:17:33.400
result = Energy.Base.Clock.Floor(needle, 0);   // 2121-12-16 21:17:33
result = Energy.Base.Clock.Floor(needle, -1);  // 2121-12-16 21:17:30
result = Energy.Base.Clock.Floor(needle, -2);  // 2121-12-16 21:17:00
result = Energy.Base.Clock.Floor(needle, -3);  // 2121-12-16 21:10:00
result = Energy.Base.Clock.Floor(needle, -4);  // 2121-12-16 21:00:00
result = Energy.Base.Clock.Floor(needle, -5);  // 2121-12-16 20:00:00
result = Energy.Base.Clock.Floor(needle, -6);  // 2121-12-16 00:00:00
result = Energy.Base.Clock.Floor(needle, -7);  // 2121-12-10 00:00:00
result = Energy.Base.Clock.Floor(needle, -8);  // 2121-12-01 00:00:00
result = Energy.Base.Clock.Floor(needle, -9);  // 2121-10-01 00:00:00
result = Energy.Base.Clock.Floor(needle, -10); // 2121-01-01 00:00:00
result = Energy.Base.Clock.Floor(needle, -11); // 2120-01-01 00:00:00
result = Energy.Base.Clock.Floor(needle, -12); // 2100-01-01 00:00:00
result = Energy.Base.Clock.Floor(needle, -13); // 2000-01-01 00:00:00
result = Energy.Base.Clock.Floor(needle, -14); // 0001-01-01 00:00:00
```

9.5 Truncate

Round down DateTime value to desired precision of seconds.

```
DateTime Energy.Base.Clock.Truncate(DateTime value, int precision)
```

Round down DateTime to whole seconds

```
DateTime Energy.Base.Clock.Truncate(DateTime value)
```

Round down TimeSpan to desired precision of seconds.

```
TimeSpan Energy.Base.Clock.Truncate(TimeSpan value, int precision)
```

9.6 HasFractionalPart

Check if DateTime value contains fractional part of seconds.

```
bool Energy.Base.Clock.HasFractionalPart(DateTime value)
```

Check if TimeSpan value contains fractional part of seconds.

```
bool Energy.Base.Clock.HasFractionalPart(TimeSpan value)
```

9.7 IsLeapYear

Sample subroutine to calculate if a year is a leap year.

```
bool Energy.Base.Clock.IsLeapYear(int year)
```

CHAPTER 10

File and directory functions

Energy.Base.File class is a collection of file and directory utility functions.

10.1 MakeDirectory

Create directory if not exists. Returns true if a directory exists or has been created.

```
bool Energy.Base.File.MakeDirectory(string path)
```

10.2 RemoveDirectory

Remove directory if exists. Returns true if directory has been removed or not exists.

```
bool Energy.Base.File.RemoveDirectory(string path, bool recursive)
```

Remove directory if exists and is empty.

```
bool Energy.Base.File.RemoveDirectory(string path)
```

10.3 GetBaseDirectory

Gets the base directory that the assembly resolver uses to probe for assemblies.

```
string Energy.Base.File.GetBaseDirectory()
```

10.4 GetBasePath

Gets the base directory that the assembly resolver uses to probe for assemblies. Return path with trailing directory separator.

```
string Energy.Base.File.GetBasePath()
```

10.5 Locate

Locate file or executable in directories from `PATH` environment variable.

If file can't be found, empty string will be returned.

```
string Energy.Base.File.Locate(string command)
```

Locate file or executable in search directories.

```
string Energy.Base.File.Locate(string command, string[] search)
```

Locate file with one of possible extensions in search directories.

```
string Energy.Base.File.Locate(string file, string[] search, string[] extension)
```

Locate file with one of possible extensions in search directories.

This version of function allows to specify lookup behaviour (iterate over directories or extensions).

```
string Energy.Base.File.Locate(string file, string[] search, string[] extension,␣
↪Energy.Enumeration.LocateBehaviour behaviour)
```

Locate file with one of possible extensions in search directory.

This version of function allows to specify lookup behaviour (iterate over directories or extensions).

```
string Energy.Base.File.Locate(string[] list, string[] search, string[] extension,␣
↪Energy.Enumeration.LocateBehaviour behaviour)
```

10.6 GetHomeDirectory

Get absolute path of home directory for current user.

Resulting path will include trailing directory separator.

```
string Energy.Base.File.GetHomeDirectory()
```

CHAPTER 11

File path helpers

Energy.Base.Path is helper class for working with file paths.

Let's look at these example paths.

```
C:\Program Files\\\.\file.txt
C:\"Program Files"\
/mnt/c/"name / slash"/'file/1'.txt
/mnt/c\\sub\\..\\/" "" "/'file\'2\''.txt
```

11.1 Split

Split path into parts. Each part will contain trailing directory separator characters.

When using without specified allowed slashes it will by default allow to use both Unix-style slash characters or Windows-style backslash character as path separator.

When using without specified quotation marks it will recognize apostrophes, quotes and backticks.

```
string[] Energy.Base.Path.Split(string path)
```

```
string[] Energy.Base.Path.Split(string path, Energy.Base.Path.SplitFormat format)
```

```
string[] Energy.Base.Path.Split(string path, string[] slashes, string[] quotes
    , bool? doublets, bool? cstyle)
```

When spliting `/home/desktop/file.txt` it will return array with four elements.

```
/
home/
desktop/
file.txt
```

For `C:\Users\Desktop\file.txt` it will work similar way.

```
C:\
Users\
Desktop\
file.txt
```

11.2 Walk

Walk through relative path and return absolute path without any dot folders.

Function will also strip repeated path separators.

```
string Energy.Base.Path.Walk(string path, string directory)
```

If second parameter is omitted, current directory will be used for relative paths.

```
string Energy.Base.Path.Walk(string path)
```

11.3 ChangeSeparator

Change any directory separator to native one for compatibility.

```
string Energy.Base.Path.ChangeSeparator(string path)
```

11.4 IsSeparator

Check if file path part is directory separator or not.

Multiple separator characters are allowed.

```
bool Energy.Base.Path.IsSeparator(string part, string[] slashes)
```

```
bool Energy.Base.Path.IsSeparator(string part, string slashes)
```

```
bool Energy.Base.Path.IsSeparator(string part)
```

11.5 StripQuotation

Strip quotation marks from file path.

Converts C:”Program Files””Dir” into C:\Program Files\Dir.

```
string Energy.Base.Path.StripQuotation(string path)
```

11.6 Environment

Get array of directories from environment variable using specifed separator character.

```
string Energy.Base.Path.Environment(string variable, char separator)
```

Get array of directories from environment variable.

```
string Energy.Base.Path.Environment(string variable)
```

Get array of directories from `PATH` environment variable.

```
string Energy.Base.Path.Environment()
```

CHAPTER 12

URL support

Objects of **Energy.Base.Url** class are used to represent URL (Uniform Resource Locator) network address parts.

12.1 Explode

Create URL object from string.

```
Energy.Base.Url Energy.Base.Url.Explode(string url)
```

You may also use implicit operator from string as in the following example.

```
Energy.Base.Url url = "http://google.com";
```

12.2 Make

Make URL address overriding parts of it.

Pass null as parameter to skip it or empty value to remove specified part from URL.

Last parameter *value* may be used to replace placeholder {0} with specified value.

```
string Energy.Base.Url.Make(string url, string scheme, string host, string port
    , string path, string query, string fragment, string user, string password
    , string value)
```

```
string Energy.Base.Url.Make(string url, string scheme, string host, int port
    , string path, string query, string fragment, string user, string password
    , string value)
```

```
string Energy.Base.Url.Make(string url, string scheme, string host, string port
    , string path, string query, string fragment, string value)
```

```
string Energy.Base.Url.Make(string url, string scheme, string host, int port
    , string path, string query, string fragment, string value)
```

```
string Energy.Base.Url.Make(string url, string scheme, string host, string port
    , string path, string query, string fragment)
```

```
string Energy.Base.Url.Make(string url, string scheme, string host, int port
    , string path, string query, string fragment)
```

12.3 Combine

Combine parts of URL together. Two or more parts are concatenated with slash.

```
string Energy.Base.Url.Combine(params string[] parts)
```

When joining function will discover if parameter part was started after question mark. Parameter parts are contatenated with ampersand character instead of slash.

```
string url;
url = Energy.Base.Url.Combine("https://www.youtube.com", "watch?v=NHCgbs3TcYg");
Console.WriteLine(url);
url = Energy.Base.Url.Combine("https://www.youtube.com", "watch?v=NHCgbs3TcYg", "t=150
↪");
Console.WriteLine(url);
```

12.4 SetHost

Set host name or address in URL.

```
string Energy.Base.Url.SetHost(string url, string host)
```

12.5 SetPort

Set port number in URL.

```
string Energy.Base.Url.SetPort(string url, string port)
```

Set port number in URL. When port number is not in range 1 .. 65535 it will be considered undefined and will be removed.

```
string Energy.Base.Url.SetPort(string url, int port)
```

12.6 SetHostAndPort

Set host name or address and port number in URL.

```
string Energy.Base.Url.SetHostAndPort(string url, string host, string port)
```

Set host name or address and port number in URL. When port number is not in range 1 .. 65535 it will be considered undefined and will be removed.

```
string Energy.Base.Url.SetHostAndPort(string url, string host, int port)
```

12.7 Overwrite

Combine two URL objects, overwriting all or only empty parts from second one.

When *all* parameter is true, values will always be overwritten with not empty parameters from second address. Otherwise, only empty values will be overwritten.

```
Energy.Base.Url Energy.Base.Url.Overwrite(Url url1, Url url2, bool all)
```

By default *all* parameter is considered to be false.

```
Energy.Base.Url Energy.Base.Url.Overwrite(Url url1, Url url2)
```

This method is also available as a method of object.

```
Energy.Base.Url Overwrite(Energy.Base.Url url, bool overwrite)
```

```
Energy.Base.Url Overwrite(Energy.Base.Url url)
```

12.8 IsEmpty

Returns true if URL is empty.

```
bool IsEmpty
```

12.9 ToString

Represent URL object as string.

```
string ToString()
```

12.10 IsUnreserved

Check if character is unreserved (allowed) character in URI according to RFC 3986.

https://tools.ietf.org/html/rfc3986

```
bool Energy.Base.Url.IsUnreserved(char c)
```

```
bool Energy.Base.Url.IsUnreserved(byte c)
```

12.11 Encode

Encode special characters for URL string, according to RFC 3986.

```
string Energy.Base.Url.Encode(string text)
```

```
string Energy.Base.Url.Encode(string text, Encoding encoding)
```

12.12 Escape

Escape all but unreserved characters for URI string with percentage codes.

```
string Energy.Base.Url.Escape(string text)
```

12.13 Unescape

Unescape previously encoded string from percentage codes.

```
string Energy.Base.Url.Unescape(string text)
```

CHAPTER 13

Anonymous types

Namespace containing short definitions of anonymous functions useful for older .NET environment.

These types are used internally by library.

13.1 String

Represents function that only output string.

```
public delegate string String();
```

13.2 State

Represents function that changes state.

```
public delegate TState State<TState>(TState input);
```

13.3 Function

Multiple definitions of generic function delegate.

```
public delegate TOut Function<TIn, TOut>(TIn input);
```

```
public delegate void Function<TIn>(TIn input);
```

```
public delegate void Function();
```

13.4 Event

Event function delegates.

```
public delegate void Event();
```

```
public delegate void Event<TEvent>(TEvent argument);
```

CHAPTER 14

Collections

Here you can find several classes for collections managed in different ways.

14.1 Table

CHAPTER 15

Connection string

Inspired by classics, a class representing a connection string to a data source. Simply, **ODBC style** *connectionString*.

15.1 Accessors

Gets or sets the value associated with the specified key. Case insensitive.

```
string this[string key]
```

15.2 Properties

Catalog name taken from one of alternatives: “Database”, “Database Name”, “Initial Catalog”.

```
string Catalog
```

Server name taken from one of alernatives: “Data Source”, “Server”, “DataSource”, “Server Name”, “Dbq”.

```
string Server
```

Protocol.

```
string Protocol
```

User name taken from one of alernatives: “User”, “User ID”.

```
string User
```

15.3 Utility functions

Quote connection string value if needed.

This method will affect on values containing space, semicolon, apostrophe or quotation mark.

```
string Energy.Base.ConnectionString.Quote(string value)
```

Strip quotes from connection string value.

```
string Energy.Base.ConnectionString.Unquote(string value)
```

Escape connection string value if needed.

```
string Energy.Base.ConnectionString.Escape(string key)
```

Represent ODBC Connection String as DSN.

```
string ToDsnString()
```

More to be add. Please fill the form if you need more.

CHAPTER 16

Regular expressions

16.1 GetGroupDescription

Get group description list from regular expression match.

```
    [0]: {0. (0) = "CHARACTER VARYING " 0:18}
    [1]: {1. (type) = "CHARACTER" 0:9}
    [2]: {2. (parameter) 0:0}
    [3]: {3. (size) 0:0}
    [4]: {4. (extra) 0:0}
    [5]: {5. (null) 0:0}
    [6]: {6. (default) 0:0}
    [7]: {7. (option) = "VARYING " 10:8}
    [8]: {8. (value) 0:0}
```

```
Energy.Base.Expression.Class.GroupDescription GetGroupDescription(string pattern,␣
↪string value, RegexOptions option)
```

```
Energy.Base.Expression.Class.GroupDescription GetGroupDescription(Regex regex, Match␣
↪match)
```

```
Energy.Base.Expression.Class.GroupDescription GetGroupDescription(string pattern,␣
↪Match match)
```

```
Energy.Base.Expression.Class.GroupDescription GetGroupDescription(Match match)
```

CHAPTER 17

Queue class

Queues are specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the list and extracted from the beginning.

Energy.Base.Queue is a generic thread-safe class which can be used to implement FIFO queues in multithreading or asynchronous environment.

Additionally, the class provides access to events when adding and removing items from the queue.

The limit is also supported. If the *Circular* option is activated after the limit has been exceeded, the oldest items are removed from the list.

17.1 Example

```
using System;

public class Program
{
        public static void Main()
        {
                var q = new Energy.Base.Queue();
                var o1 = "123";
                var o2 = "abc";
                // insert object into queue by calling Push()
                q.Push(o1);
                q.Push(o2);
                q.Push(o1);
                q.Push(o2);
                q.Push(o2);
                q.Push(o2);
                object o;
                // retreive object from queue by calling Pull()
                while (null != (o = q.Pull()))
                {
```

(continues on next page)

(continued from previous page)

```
                        Console.WriteLine(o);
                }
        }
}
```

Open in **dotnetfiddle.net**

17.2 Properties

Numer of elements in queue.

```
int Count
```

Check if queue is empty.

```
bool IsEmpty
```

Limit number of items in queue.

```
int Limit
```

17.3 Ring mode

Makes internal buffer work like circular buffer. When this option is set, the oldest items are removed from the list when limit has been exceeded.

```
bool Ring
```

Example of circular buffer use.

```
Energy.Base.Queue<string> queue = new Energy.Base.Queue<string>();
queue.Ring = true;
queue.Limit = 2;
queue.Push("A");
queue.Push("B");
queue.Push("C");
// number of element will be 2 because both limitand circular option are set
string value;
value = queue.Pull();
System.Diagnostics.Debug.WriteLine(value); // "B"
value = queue.Pull();
System.Diagnostics.Debug.WriteLine(value); // "C"
// queue is now empty, so next element will be null
value = queue.Pull();
queue.Ring = false;
bool success;
success = queue.Push("A"); // true
success = queue.Push("B"); // true
success = queue.Push("C"); // false because limit is reached
```

17.4 Functions

Put element at the end of queue. If limit is reached, function will return false and element will not be put at the end of the queue unless Ring option is set.

```
bool Energy.Base.Text.Push(T item)
```

Put array of elements at the end of queue.

```
bool Energy.Base.Text.Push(T[] array)
```

Take first element from queue, remove it from queue, and finally return. If queue is empty, function will return null.

```
T Energy.Base.Text.Pull()
```

Take number of elements from queue, remove them and return array of elements taken. **Pull(0)** will return all elements from queue and empty it.

```
T[] Energy.Base.Text.Pull(int count)
```

Take element from queue with specified time limit to wait for new item to come. It will pause invoking thread as it is expected to do so.

```
T Energy.Base.Text.Pull(double timeout)
```

Put element back to queue, at begining. This element will be taken first.

```
void Energy.Base.Text.Back(T item)
```

Put array of elements back to queue, at begining. These elements will be taken first.

```
void Energy.Base.Text.Back(T[] list)
```

Delete last element from queue and return it.

```
T Energy.Base.Text.Chop()
```

Delete number of last elements from queue and return them.

```
T[] Energy.Base.Text.Chop(int count)
```

17.5 Events

Event fired when Push() is called and element was added to the queue.

```
event Energy.Base.Anonymous.Event OnPush
```

Event fired when Pull() is called and element was taken from the queue.

```
event Energy.Base.Anonymous.Event OnPull
```

Event fired when Back() is called and element was put back to the queue.

```
event Energy.Base.Anonymous.Event OnBack
```

Event fired when Chop() is called and element was deleted from the queue.

```
event Energy.Base.Anonymous.Event OnChop
```

CHAPTER 18

XML support

18.1 Serialize

Serialize object to XML.

Object class must implement **IXmlSerializable** interface.

```
string Energy.Base.Xml.Serialize(object data, string root, string space)
```

```
string Energy.Base.Xml.Serialize(object data, string root)
```

```
string Energy.Base.Xml.Serialize(object data)
```

18.1.1 Example

```
string xml = Energy.Base.Xml.Serialize(myObject, "Root", "org.example.xns");
```

18.2 Deserialize

Deserialize object from XML, root alternatives allowed.

```
object Energy.Base.Xml.Deserialize(string content, Type type, string[] root, string␣
↪space)
```

```
object Energy.Base.Xml.Deserialize(string content, Type type, string root, string␣
↪space)
```

```
object Energy.Base.Xml.Deserialize(string content, Type type, string root)
```

```
object Energy.Base.Xml.Deserialize(string content, Type type)
```

Generic XML deserialization methods are also available.

```
TDeserialize Energy.Base.Xml.Deserialize<TDeserialize>(string content, Type type,␣
↪string[] root, string space)
```

```
TDeserialize Energy.Base.Xml.Deserialize<TDeserialize>(string content)
```

18.3 ExtractRoot

Extract root element from XML.

```
string Energy.Base.Xml.ExtractRoot(string xml)
```

18.4 ExtractRootShort

Extract root element without namespace from XML.

```
string Energy.Base.Xml.ExtractRootShort(string xml)
```

18.5 Encode

Encode special characters with valid XML entities.

Only ASCII control codes and XML special characters will be encoded. All other valid UTF-8 characters will remain untouched.

Control characters like new line, carriage return, and tab will not be encoded either.

```
string Energy.Base.Xml.Encode(string text)
```

Encode special characters with valid XML entities.

When encoding parameter is set to Encoding.UTF-8 then only ASCII control codes and XML special characters will be encoded. All other valid UTF-8 characters will remain untouched.

Control characters like new line, carriage return, and tab will not be encoded either.

When encoding parameter is set to Encoding.ASCII then additionaly all characters with codes higher that 126 will be encoded as character entities.

When encoding parameter is set to Encoding.Unicode then Unicode surrogate pairs (i.e. emoji) will also be encoded as character entities.

```
string Energy.Base.Xml.Encode(string text, Encoding encoding)
```

18.6 Decode

Decode named or numeric character XML entities with corresponding characters.

```
string Energy.Base.Xml.Decode(string text)
```

CHAPTER 19

JSON functions

19.1 Escape

Escape special characters for JSON value.

```
Backspace is replaced with '\b'.
Form feed is replaced with '\f'.
Newline is replaced with '\n'.
Carriage return is replaced with '\r'.
Tab is replaced with '\t'.
Double quote is replaced with '\"'.
Backslash is replaced with '\\'.
```

Escaped text must be surrounded with double quotes.

```
string Energy.Base.Json.Escape(string text)
```

19.2 Unescape

Strip backslashes from previously escaped value.

```
string Energy.Base.Json.Unescape(string text)
```

19.3 Quote

Represents text in double quotes and escapes special characters.

Null strings are represented as single word “null”.

```
string Energy.Base.Json.Quote(string text)
```

19.4 Strip

Strip double quotes and unescapes string.

```
string Energy.Base.Json.Strip(string text)
```

CHAPTER 20

Binary and byte functions

Energy.Base.Bit class is a collection of binary utility functions.

20.1 Compare

Compare byte arrays.

```
int Energy.Base.Bit.Compare(byte[] left, byte[] right)
```

20.2 Reverse

Reverse order of bytes.

Exchange lower byte with higher one

```
ushort Energy.Base.Bit.Reverse(ushort value)
```

```
uint Energy.Base.Bit.Reverse(uint value)
```

20.3 Endianess

Take up to two 16-bit unsigned words and return them as 32-bit unsigned word.

MSB / Big-Endian.

```
UInt32 Energy.Base.Bit.GetUInt32MSB(params UInt16[] array)
```

Take up to two 16-bit unsigned words and return them as 32-bit unsigned word.

LSB / Little-Endian.

```
UInt32 Energy.Base.Bit.GetUInt32LSB(params UInt16[] array)
```

20.4 NOT

Perform bitwise NOT operation on every byte in array.

```
byte[] Energy.Base.Bit.Not(byte[] array)
```

20.5 OR

Perform bitwise OR operation on every byte in array by second array.

Second array is treated as ring buffer when shorter than first one.

```
byte[] Energy.Base.Bit.Or(byte[] one, byte[] two)
```

20.6 AND

Perform bitwise AND operation on every byte in array by second array.

Second array is treated as ring buffer when shorter than first one.

```
byte[] Energy.Base.Bit.And(byte[] one, byte[] two)
```

20.7 XOR

Perform bitwise XOR operation on every byte in array by second array.

Second array is treated as ring buffer when shorter than first one.

```
byte[] Energy.Base.Bit.Xor(byte[] one, byte[] two)
```

20.8 ROL

Rotate bits left in an array by given bit count.

When negative number of bits is given, right rotation will be performed instead.

```
byte[] Energy.Base.Bit.Rol(byte[] array, int count)
```

20.9 ROR

Rotate bits right in an array by given bit count.

When negative number of bits is given, left rotation will be performed instead.

```
byte[] Energy.Base.Bit.Ror(byte[] array, int count)
```

CHAPTER 21

Byte array builder

Use ByteArrayBuilder to build byte arrays with MemoryStream.

This class is not thread safe for performance reasons.

You must use your own locking mechanism to achieve thread safety.

Objects of **Energy.Base.ByteArrayBuilder** works like **streams** supporting seeking from the beginning of the stream. You may also call *Rewind()* or *Tail()*.

```
Energy.Base.ByteArrayBuilder b = new Energy.Base.ByteArrayBuilder();
b.WriteByte(1);
b.WriteByte(2);
b.Rewind();
t = b.ReadByte();
b.Tail();
b.WriteByte(3);
b.Seek(1);
t = b.ReadByte();
```

Automatic size of data for primitive types.

```
var value = new int[] { 1, 2, 3, 4 };
var result = Energy.Base.ByteArrayBuilder.ToByteArray(value);
var expect = new byte[] { 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0, 4 };
```

Energy.Base.ByteArrayBuilder also supports Base64 format.

```
Energy.Base.ByteArrayBuilder bb;
bb = new Energy.Base.ByteArrayBuilder();
string s1 = "TWljcm9zb2Z0IFdpbmRvd3M=";
bb.WriteBase64(s1);
bb.Rewind();
string s2 = bb.ReadString();
// s2 equals "Microsoft Windows"
bb.Clear();
```

(continues on next page)

(continued from previous page)

```
string s3 = "Gęś";
bb.WriteString(s3);
bb.Rewind();
string s4 = bb.ReadBase64();
// s4 equals "R8SZxZs="
```

CHAPTER 22

Compression

22.1 Deflate

Compress using deflate algorithm.

```
byte[] Energy.Base.Compression.Compress(byte[] data)
```

Decompress using deflate algorithm.

```
byte[] Energy.Base.Compression.Decompress(byte[] data)
```

22.2 GZip

Compress using gzip algorithm.

```
byte[] Energy.Base.Compression.Compress(byte[] data)
```

Decompress using gzip algorithm.

```
byte[] Energy.Base.Compression.Decompress(byte[] data)
```

Not much... yet... :-)

CHAPTER 23

Hexadecimal

Functions that support hexadecimal numeral system conversions.

23.1 ArrayToHex

Convert byte array to hexadecimal string.

```
string Energy.Base.Hex.ArrayToHex(byte[] array)
```

```
string Energy.Base.Hex.ArrayToHex(byte[] array, string space)
```

23.2 HexToArray

Convert hexadecimal string to byte array.

This is strict version of conversion function.

Hexadecimal string should contain only digits and small or upper letters A-F. Any other character is treated as zero.

```
byte[] Energy.Base.Hex.HexToArray(string hex)
```

This version of conversion function allows to use whitespace characters.

```
byte[] Energy.Base.Hex.HexToArray(string hex, bool ignoreWhite)
```

This version of conversion function allows to use prefixes (like “0x” or “$”) and whitespace characters.

```
byte[] Energy.Base.Hex.HexToArray(string hex, bool ignoreWhite, string[] prefix)
```

```
var b = Energy.Base.Hex.HexToArray("0x 12 33", true, new string[] { "0x" });
```

23.3 BinToHex

Convert binary string to hexadecimal string.

```
string Energy.Base.Hex.BinToHex(string bin)
```

23.4 HexToBin

Convert hexadecimal string to binary string.

Note that hexadecimal "0" will be represented with leading zeroes as "0000" in binary. Resulting binary string will always have a length divisible by 4.

Works also when hexadecimal string starts with "0x" or "$".

```
string Energy.Base.Hex.HexToBin(string hex)
```

23.5 HexToByte

Convert hexadecimal string to byte value (System.Byte).

```
byte Energy.Base.Hex.HexToByte(string hex)
```

23.6 RemovePrefix

Remove leading prefix "0x", "0X" or "$" from hexadecimal string.

```
string Energy.Base.Hex.RemovePrefix(string hex)
```

23.7 ByteToPrintable

Convert byte to printable character.

All non-ASCII characters will be represented as dot character. Bytes 0, 128 and 255 will be represented as space.

```
char Energy.Base.Hex.ByteToPrintable(byte b)
```

CHAPTER 24

Geographic coordinate system

24.1 Point

Location point.

```
class Energy.Base.Geo.Point
{
    [XmlElement("Y")]
    [DefaultValue(0)]
    public double Latitude;

    [XmlElement("X")]
    [DefaultValue(0)]
    public double Longitude;

    [XmlElement("Z")]
    [DefaultValue(0)]
    public double Altitude;
}
```

24.2 GetDistance

Get earth distance between two locations in kilometers.

```
double Energy.Base.Geo.GetDistance(Energy.Base.Geo.Point location1, Energy.Base.Geo.
↪Point location2)
```

Get distance between two locations using globe radius value.

```
double Energy.Base.Geo.GetDistance(Energy.Base.Geo.Point location1, Energy.Base.Geo.
↪Point location2, double R)
```

CHAPTER 25

Naming convention

These functions may be helpful to write convention tests.

Please keep in mind that when validating, leading special characters like `@`, _ or `$` are not allowed as identifiers must start with letters only.

25.1 CamelCase

Return words capitalized with first word in lower case.

`camelCase`

```
string Energy.Base.Naming.CamelCase(string[] words)
```

Check if text is valid identifier written in camelCase.

```
bool Energy.Base.Naming.IsCamelCase(string text)
```

25.2 CobolCase

Return words upper case, separated with hyphen character.

`COBOL-CASE`

```
string Energy.Base.Naming.CobolCase(string[] words)
```

Check if text is valid identifier written in COBOL-CASE.

```
bool Energy.Base.Naming.IsCobolCase(string text)
```

25.3 ConstantCase

Return words upper case, separated with underscore character.

`CONSTANT_CASE`

```
string Energy.Base.Naming.ConstantCase(string[] words)
```

Check if text is valid identifier written in CONSTANT_CASE.

```
bool Energy.Base.Naming.IsConstantCase(string text)
```

25.4 DashCase

Return words lower case, separated with hyphen character.

`dash-case`

This style is often used in URLs to give more human-readable look.

Also known as kebab-case or hyphen-case.

```
string Energy.Base.Naming.DashCase(string[] words)
```

Check if text is valid identifier written in dash-case.

```
bool Energy.Base.Naming.IsDashCase(string text)
```

25.5 HyphenCase

Return words lower case, separated with hyphen character.

`hyphen-case`

This style is often used in URLs to give more human-readable look.

Also known as dash-case or kebab-case.

```
string Energy.Base.Naming.HyphenCase(string[] words)
```

Check if text is valid identifier written in hyphen-case.

```
bool Energy.Base.Naming.IsHyphenCase(string text)
```

25.6 KebabCase

Return words lower case, separated with hyphen character.

`kebab-case`

```
string Energy.Base.Naming.KebabCase(string[] words)
```

Check if text is valid identifier written in kebab-case.

```
bool Energy.Base.Naming.IsKebabCasee(string text)
```

25.7 PascalCase

Return words capitalized.

`PascalCase`

```
string Energy.Base.Naming.PascalCase(string[] words)
```

Check if text is valid identifier written in PascalCase.

```
bool Energy.Base.Naming.IsPascalCase(string text)
```

25.8 SnakeCase

Return words lower case, joined with underscore character.

`snake_case`

Also known as underscore_case.

```
string Energy.Base.Naming.SnakeCase(string[] words)
```

Check if text is valid identifier written in camelCase.

```
bool Energy.Base.Naming.IsSnakeCase(string text)
```

25.9 TrainCase

Return words capitalized and separated by hyphen character.

`Train-Case`

```
string Energy.Base.Naming.TrainCase(string[] words)
```

Check if text is valid identifier written in Train-Case.

```
bool Energy.Base.Naming.IsTrainCase(string text)
```

25.10 UnderscoreCase

Return words lower case, joined with underscore character.

`underscore_case`

Also known as snake_case.

```
string Energy.Base.Naming.UnderscoreCase(string[] words)
```

Check if text is valid identifier written in underscore_case.

```
bool Energy.Base.Naming.IsUnderscoreCase(string text)
```

CHAPTER 26

Enumerations

26.1 BasicType

Basic data type.

Generalisation of the type of data used in data exchange systems, SQL databases, etc. The assumption is to use as few variants as possible, so there is no single character type. Instead, you should use a text type with a character limit. If necessary, a single character can be treated as an integer.

An integer is both an integer and a negative number, as well as a real number. Integers are of course also general numbers.

Date and time are represented by 3 values depending on whether it is date (YYYY-MM-DD), time (hh:mm:ss) or full time stamp (YYYY-MM-DD hh:mm:ss).

```
enum Energy.Enumeration.BasicType
{
    /// <summary>
    /// Text value
    /// </summary>
    Text,

    /// <summary>
    /// Any value which is number
    /// </summary>
    Number,

    /// <summary>
    /// Integer numbers only
    /// </summary>
    Integer,

    /// <summary>
    /// True / False
    /// </summary>
```

(continues on next page)

(continued from previous page)

```
    Bool,

    /// <summary>
    /// Date
    /// </summary>
    Date,

    /// <summary>
    /// Time
    /// </summary>
    Time,

    /// <summary>
    /// Date and time
    /// </summary>
    Stamp,
}
```

26.2 BooleanStyle

Style of representing boolean values.

Used by **Energy.Cast.BoolToString()**.

```
enum Energy.Enumeration.BooleanStyle
{
    /// <summary>
    /// 0/1
    /// </summary>
    B,

    /// <summary>
    /// X for true
    /// </summary>
    X,

    /// <summary>
    /// V for true
    /// </summary>
    V,

    /// <summary>
    /// Y/N
    /// </summary>
    Y,

    /// <summary>
    /// T/F
    /// </summary>
    T,

    /// <summary>
    /// Yes/No
    /// </summary>
```

(continues on next page)

(continued from previous page)

```
    /// <remarks>Localised</remarks>
    YesNo,

    /// <summary>
    /// True/False
    /// </summary>
    /// <remarks>Localised</remarks>
    TrueFalse,

    /// <summary>
    /// 0/1
    /// </summary>
    /// <remarks>Localised</remarks>
    Bit = B,

    /// <summary>
    /// Y/N
    /// </summary>
    /// <remarks>Localised</remarks>
    YN = Y,

    /// <summary>
    /// T/F
    /// </summary>
    /// <remarks>Localised</remarks>
    TF = T,
}
```

26.3 MultipleBehaviour

Selection of duplicates behaviour.

Specifies behaviour for selecting one element from multiple duplicates.

The default behavior is to overwrite the value and thus taking the last value.

```
enum Energy.Enumeration.MultipleBehaviour
{
    /// <summary>
    /// Unspecified behaviour.
    /// </summary>
    None,

    /// <summary>
    /// Take last from duplicates.
    /// Setting value will overwrite element if exists.
    /// </summary>
    Last,

    /// <summary>
    /// Take first from duplicates.
    /// Value may be set only once. It will be ignored if element exists.
    /// </summary>
    First,
}
```

26.4 RoundingMethod

Method of rounding numbers.

```
enum Energy.Enumeration.RoundingMethod
{
    None = Floor,

    /// <summary>
    /// Standard method of rounding (HalfUp)
    /// </summary>
    Standard = HalfUp,

    /// <summary>
    /// Round down
    /// </summary>
    Floor = 0,

    /// <summary>
    /// Round up
    /// </summary>
    Ceil = 0,

    /// <summary>
    /// Half Round Up
    /// </summary>
    HalfUp = 4,

    /// <summary>
    /// Half Round Down
    /// </summary>
    HalfDown = 5,

    /// <summary>
    /// Round to Even (Banker's Rounding)
    /// </summary>
    ToEven = 2,

    /// <summary>
    /// Round to Odd.
    /// </summary>
    ToOdd = 3,
}
```

26.4.1 Half Round Up

This method is commonly used.

> *However, some programming languages (such as Java, Python) define their half up as round half away from zero here.*

http://en.wikipedia.org/wiki/Rounding

```
Energy.Base.Enumeration.HalfUp
Energy.Base.Enumeration.Standard
```

26.4.2 Half Round Down

Example: 7.6 rounds up to 8, 7.5 rounds down to 7, 7.4 rounds down to 7.

Anyone remembers the “5/4” mode in vintage calculators?


```
Energy.Base.Enumeration.HalfDown
```

26.4.3 Round to Even (Banker’s Rounding)

Example: 7.5 rounds up to 8 (because 8 is an even number) but 6.5 rounds down to 6 (because 6 is an even number).

```
Energy.Base.Enumeration.ToEven
```

26.4.4 Round to Odd

Example: 7.5 rounds down to 7 (because 7 is an odd number) but 6.5 rounds up to 7 (because 7 is an odd number).

```
Energy.Base.Enumeration.ToOdd
```

CHAPTER 27

Tilde coloring engine

27.1 Introduction

Simple elegant text coloring engine for console programs. Based on **Empty Page #0** color engine.

Color may be specified by its number or name surrounded by tilde (~) character. One tilde followed by a character other than a letter or number or a fragment consisting of two or more tilde characters will not be formatted.

Colors that can be used are the same as on a standard command console. There are 15 different colours plus black.

Dark colour identifiers are preceded by the letter 'd' (dark). In a similar way, bright-colour identifiers can be preceded by the letter 'l' (light).

Color changes one by one are silently ignored. Only the last defined color will be used.

Current color is remembered before writing color text and restored after writing. Special *color* ~0~ may be used to force setting back original color inside text.

27.2 Examples

```
    Energy.Core.Tilde.WriteLine("~yellow~Hello, ~cyan~world~white~!");
```

```
Hello, world!
```

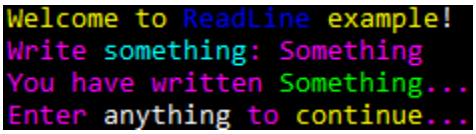
```
    Energy.Core.Tilde.Write(" ~1~{1}~2~{2}~3~{3}~4~{4}~5~{5}~6~{6} ", null
        , 1, 2, 3, 4, 5, 6);
```

```
 123456
```

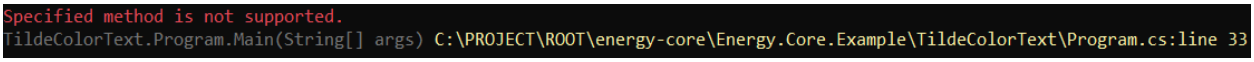
```
    Energy.Core.Tilde.WriteLine("You can use ~`~yellow~`~ to mark text ~yellow~yellow~
↪0~.");
```

```
You can use ~yellow~ to mark text yellow.
```

```
    Energy.Core.Tilde.WriteLine("~yellow~Welcome to ~blue~ReadLine ~yellow~example~
↪white~!");
    Console.ForegroundColor = ConsoleColor.Magenta;
    Energy.Core.Tilde.Write("Write ~c~something~0~: ~magenta~");
    while (true)
    {
        string input = Energy.Core.Tilde.ReadLine();
        if (input == null)
        {
            System.Threading.Thread.Sleep(500);
            continue;
        }
        else
        {
            Energy.Core.Tilde.WriteLine("You have written ~green~{0}~0~...", input);
            break;
        }
    }
    Energy.Core.Tilde.Pause();
```




```
try
{
    throw new NotSupportedException();
}
catch (Exception exception)
{
    Energy.Core.Tilde.WriteException(exception, true);
}
```



27.3 Pause

Writes out pause text and waits for user to input anything by reading line from console.

```
void Energy.Core.Tilde.Pause()
```

27.4 Break

Write out one break line and set default text color.

```
void Energy.Core.Tilde.Break()
```

Write out empty lines and set default text color.

```
void Energy.Core.Tilde.Break(int count)
```

Write out ruler line surrounded by empty lines.

```
void Energy.Core.Tilde.Break(int padding, string line)
```

27.5 ReadLine

Read line from console if available. Does not wait for user to enter anything so may be useful in loops. Thread safe. Returns null if user did not press Enter key.

```
string Energy.Core.Tilde.ReadLine()
```

```
Energy.Core.Tilde.WriteLine("~yellow~Welcome to ~blue~ReadLine ~yellow~example~white~!
↪");
Console.ForegroundColor = ConsoleColor.Magenta;
Energy.Core.Tilde.Write("Write ~c~something~0~: ~magenta~");
while (true)
{
    string input = Energy.Core.Tilde.ReadLine();
    if (input == null)
    {
        Thread.Sleep(500);
        continue;
    }
    else
    {
        Energy.Core.Tilde.WriteLine("You have written ~green~{0}~0~..."
            , input);
        break;
    }
}
```

```
Welcome to ReadLine example!
Write something: Nothing
You have written Nothing...
Enter anything to continue...
```

27.6 Input

Write out prompt message and wait for input string. If empty string is read from console, function will return *defaultValue* parameter value.

If message contains placeholder **{0}**, it will be replaced with *defaultValue* parameter value.

```
public static string Input(string message, string defaultValue)
```

27.7 Exception

Write out exception message with optional stack trace.

```
public static void Exception(Exception exception, bool trace)
```

```
Specified method is not supported.
TildeColorText.Program.Main(String[] args) C:\PROJECT\ROOT\energy-core\Energy.Core.Example\TildeColorText\Program.cs:line 33
```

27.8 Strip

Strip tildes from text.

```
public static string Strip(string text)
```

27.9 Escape

Escape string which may contain tilde characters.

```
public static string Escape(string text)
```

27.10 Length

Return total length of tilde string. Doesn't count tilde control strings.

```
public static int Length(string example)
```

27.11 Color

Tilde coloring engine console color string table.

```
public class Color
{
    public static string DarkBlue = "~1~";
    public static string DarkGreen = "~2~";
    public static string DarkCyan = "~3~";
    public static string DarkRed = "~4~";
    public static string DarkMagenta = "~5~";
    public static string DarkYellow = "~6~";
    public static string Gray = "~7~";
    public static string DarkGray = "~8~";
    public static string Blue = "~9~";
    public static string Green = "~10~";
    public static string Cyan = "~11~";
    public static string Red = "~12~";
    public static string Magenta = "~13~";
    public static string Yellow = "~14~";
    public static string White = "~15~";
    public static string Black = "~16~";
}
```

CHAPTER 28

Tilde color table

Numeric	Long	Short	
~0~	~default~		Default color
~1~	~darkblue~	~db~	Dark blue
~2~	~darkgreen~	~dg~	Dark green
~3~	~darkcyan~	~dc~	Dark cyan
~4~	~darkred~	~dr~	Dark red
~5~	~darkmagenta~	~dm~	Dark magenta
~6~	~darkyellow~	~dy~	Dark yellow
~7~	~gray~	~s~	Gray / Silver
~8~	~darkgray~	~ds~	Dark gray / Dark silver
~9~	~blue~	~b~	Blue
~10~	~green~	~g~	Dark yellow
~11~	~cyan~	~c~	Cyan
~12~	~red~	~r~	Red
~13~	~magenta~	~m~	Magenta
~14~	~yellow~	~y~	Yellow
~15~	~white~	~w~	White
~16~	~black~	~k~	Black / Carbon

CHAPTER 29

Web functions

29.1 Certificate validation

In some cases certificate validation has to be switched off when using self signed certificates is SSL connectivity.

You may use *Energy.Core.Web.IgnoreCertificateValidation* property to change this setting programatically.

```
public static bool IgnoreCertificateValidation
```

29.2 REST

Formal methods are defined "HEAD", "GET", "PUT", "POST", "PATCH", "DELETE", "OPTIONS".

All REST methods use *Energy.Base.Http.Request* and *Energy.Base.Http.Response* classes which represents web request and response.

29.2.1 GET

Perform GET and return response from URL.

```
public static Energy.Base.Http.Response Get(string url)
```

```
public static Energy.Base.Http.Response Get(string url, string acceptType)
```

```
public static Energy.Base.Http.Response Get(Energy.Base.Http.Request request)
```

29.2.2 POST

Perform POST and return response from HTTP request.

```
public static Energy.Base.Http.Response Post(Energy.Base.Http.Request request)
```

Perform POST and return response from URL.

```
public static Energy.Base.Http.Response Post(string url, object body, string␣
↪contentType, string[] headers, string acceptType)
```

A little too many parameters, right?

```
public static Energy.Base.Http.Response Post(string url, object body, string␣
↪contentType, string[] headers)
```

```
public static Energy.Base.Http.Response Post(string url, object body, string␣
↪contentType)
```

```
public static Energy.Base.Http.Response Post(string url, object body)
```

```
public static Energy.Base.Http.Response Post(string url)
```

29.2.3 PUT

Perform PUT and return response from URL.

```
public static Energy.Base.Http.Response Put(string url, object body, string␣
↪contentType, string[] headers)
```

```
public static Energy.Base.Http.Response Put(string url)
```

```
public static Energy.Base.Http.Response Put(string url, object body, string␣
↪contentType)
```

```
public static Energy.Base.Http.Response Put(string url, object body)
```

29.2.4 PATCH

Perform PATCH and return response from URL.

```
public static Energy.Base.Http.Response Patch(Energy.Base.Http.Request request)
```

```
public static Energy.Base.Http.Response Patch(string url, object body, string␣
↪contentType, string[] headers, string acceptType)
```

```
public static Energy.Base.Http.Response Patch(string url, object body, string␣
↪contentType)
```

```
public static Energy.Base.Http.Response Patch(string url, object body)
```

```
public static Energy.Base.Http.Response Patch(string url)
```

29.2.5 HEAD

Perform HEAD and return response from URL.

```
public static Energy.Base.Http.Response Head(Energy.Base.Http.Request request)
```

```
public static Energy.Base.Http.Response Head(string url)
```

```
public static Energy.Base.Http.Response Head(string url, out string[] responseHeaders)
```

29.2.6 DELETE

Perform DELETE and return response from URL.

```
public static Energy.Base.Http.Response Delete(Energy.Base.Http.Request request)
```

```
public static Energy.Base.Http.Response Delete(string url)
```

29.2.7 OPTIONS

Perform OPTIONS and return response from URL.

```
public static Energy.Base.Http.Response Options(Energy.Base.Http.Request request)
```

```
public static Energy.Base.Http.Response Options(string url)
```

CHAPTER 30

Database access

Energy.Core.Source provides access to any SQL database source. It uses **ADO.NET** so any compatible driver can be used.

How this can be helpful?

It gives bunch of primitive functions that can be done using SQL database. That means executing queries, reading data from them, and maintain a connection if needed. This class also provides error handling.

Higher levels of database abstractions like models or ORM mapping are not concerned here.

30.1 Connection

Create **Energy.Source.Connection** object to enable database access layer operations.

You need to specify vendor class which implements **IDbConnection** interface from **ADO.NET**.

```
Energy.Source.Connection(Type vendor)
```

```
Energy.Source.Connection(Type vendor, string connectionString)
```

```
Energy.Source.Connection<Type>()
```

```
Energy.Source.Connection<Type>(string connectionString)
```

30.1.1 Vendor

30.1.2 Example

```
Energy.Source.Connection<MySql.Data.MySqlClient.MySqlConnection> db;
db = new Energy.Source.Connection<MySql.Data.MySqlClient.MySqlConnection>();
db.ConnectionString = @"Server=127.0.0.1;Database=test;Uid=test;Pwd=test;";
```

30.2 Execute

Execute SQL query.

For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command.

For all other types of statements, the return value is -1.

On error, return value is -2.

```
int Execute(string query, out string error)
```

```
int Execute(string query)
```

30.3 Load

Load data from query into DataTable.

```
DataTable Load(string query, out string error)
```

```
DataTable Load(string query)
```

30.4 Read

Read query results into DataTable.

This function will populate values in a loop using IDataReader.

```
DataTable Read(string query, out string error)
```

```
DataTable Read(string query)
```

30.5 Fetch

Fetch query results into Energy.Base.Table.

```
Energy.Base.Table Fetch(string query, out string error)
```

```
Energy.Base.Table Fetch(string query)
```

30.6 Scalar

Execute SQL query and read scalar value as a result.

```
object Scalar(string query, out string error)
```

```
object Scalar(string query)
```

```
T Scalar<T>(string query, out string error)
```

```
T Scalar<T>(string query)
```

30.7 Clone

Return copy of object.

```
object Clone()
```

```
Energy.Source.Connection Copy()
```

30.8 Persistent

If you want to limit connections to your database source you may turn on pertistent option by setting **Persistent** to **true**.

It's not normally required to do that so at database access layer, but you still might need it in some particular cases.

Only one **ADO.NET** connection object will be used through multiple executions. This way concurrent SQL executions will have to wait which is acquired with standard .NET locking mechanism. That will have performance impact for sure, although it might not be an issue.

Useful when using in-memory database connections created per connection.

Running hundreds or thousands of operations on the database may cause performance results to be different when using persistent option.

30.9 Events

Event fired when vendor connection object is created by Activator.

```
event EventHandler OnCreate;
```

Event fired when vendor connection is open.

```
event EventHandler OnOpen;
```

Event fired when vendor connection was closed.

```
event EventHandler OnClose;
```

30.10 GetErrorText

Get error text.

```
string GetErrorText()
```

CHAPTER 31

Benchmarking

31.1 Loop

Repeat action for a specified time and return number of iterations done during the specified time. If time was not specified function will result -1. If no operation has been performed before the specified time has elapsed, the function will return a zero value.

31.1.1 Prototype

```
static int Energy.Core.Benchmark.Loop(Energy.Base.Anonymous.Function, TimeSpan)
```

31.1.2 Example

```
int count = Energy.Core.Benchmark.Loop(() =>
{
    int next = r.Next();
    string text = next.ToString();
    int.TryParse(text, out next);
}, TimeSpan.FromSeconds(1.0));
Energy.Core.Tilde.WriteLine("Done ~m~default ~w~TryParse~0~ " + count);
```

31.2 Profile

Perform a profiling operation by launching the action the specified number of times and returning the result of the profiling.

31.2.1 Prototype

```
static Energy.Core.Benchmark.Result Profile(Energy.Base.Anonymous.Function, int, int)
static Energy.Core.Benchmark.Result Profile(Energy.Base.Anonymous.Function, int,␣
↪string)
static Energy.Core.Benchmark.Result Profile(Energy.Base.Anonymous.Function, int, int,␣
↪string)
static Energy.Core.Benchmark.Result Profile(Energy.Base.Anonymous.Function, int)
static Energy.Core.Benchmark.Result Profile(Energy.Base.Anonymous.Function)
```

31.2.2 Example

```
Energy.Core.Benchmark.Result benchmark;
benchmark = Energy.Core.Benchmark.Profile(() =>
{
    long m = 0;
    for (int i = 0, n = 0; i < 100000; i++)
    {
        string s = i.ToString();
        if (Energy.Base.Text.TryParse(s, out n))
        {
            m += n;
        }
    }
}, 3, 1, "SumTryParse");
Energy.Core.Tilde.WriteLine(benchmark.ToString());
```

31.2.3 Output

```
Garbage collected in 0.002 s
Time taken by SumTryParse in 3 iterations 0.047 s
Average time of execution 0.024 s
```

CHAPTER 32

Text editor

32.1 Editor

Text editor class.

```
Energy.Core.Text.Editor editor;
// create editor object
editor = new Energy.Core.Text.Editor();
// or take global default
editor = Energy.Core.Text.Editor.Default;
```

32.1.1 InsertBeforeFirstLine

Insert text before first line.

```
string InsertBeforeFirstLine(string text, string line)
```

32.1.2 AppendAfterFirstLine

Append text after first line.

```
string AppendAfterFirstLine(string text, string line)
```

32.1.3 InsertBeforeSecondLine

Insert text before second line.

```
string InsertBeforeSecondLine(string text, string line)
```

32.1.4 InsertBeforeLastLine

Insert text before last line.

```
string InsertBeforeLastLine(string text, string line)
```

32.1.5 AppendAfterLastLine

Append text after last line.

```
string AppendAfterLastLine(string text, string line)
```

32.1.6 GetLastLine

Get last line of text.

```
string GetLastLine(string text)
```

32.1.7 EnsureNewLineAtEnd

Ensure text ends with newline. Add newline string to the end if not included even if empty.

Works with multiple newline strings from **Energy.Base.Text.NEWLINE_ARRAY**.

```
string EnsureNewLineAtEnd(string text)
```

32.1.8 AppendAfterLastLine

Append text after last line.

```
string AppendAfterLastLine(string text, string line)
```

32.1.9 ConvertNewLine

Convert new line delimiter to specified one.

```
string ConvertNewLine(string text, string newLine)
```

Convert newline delimiter to environment default. Value of constant **Energy.Base.Text.NL** is used.

```
string ConvertNewLine(string text)
```

CHAPTER 33

Variable syntax parser

33.1 Example

```
string interpolatedText;
interpolatedText = Energy.Core.Syntax.Default.Parse("{{HOUR_12}}:{{MM}}:{{SS}}.{{MS}}
↪{{AM}}");
```

CHAPTER 34

Memory management

34.1 GetCurrentMemoryUsage

Get current memory usage.

```
long Energy.Core.Memory.GetCurrentMemoryUsage()
```

34.2 Clear

Clear MemoryStream object.

```
void Energy.Core.Memory.Clear(MemoryStream stream)
```

CHAPTER 35

Program utilities

35.1 GetAssembly

Get current assembly from GetExecutingAssembly or GetCallingAssembly.

This function will not throw any exception, returning null on any error.

```
System.Reflection.Assembly Energy.Core.Program.GetAssembly()
```

35.2 GetExecutionFile

Get execution file location from current working assembly (calling or executing).

```
System.Reflection.Assembly Energy.Core.Program.GetExecutionFile()
```

35.3 GetExecutionDirectory

Get execution directory from the assembly location.

Resulting directory will contain trailing path separator.

```
System.Reflection.Assembly Energy.Core.Program.GetExecutionDirectory(System.
↪Reflection.Assembly assembly)
```

```
System.Reflection.Assembly Energy.Core.Program.GetExecutionDirectory()
```

35.4 GetCommandName

Get short command name from assembly location.

```
System.Reflection.Assembly Energy.Core.Program.GetCommandName(System.Reflection.
↪Assembly assembly)
```

```
System.Reflection.Assembly Energy.Core.Program.GetCommandName()
```

35.5 SetLanguage

Set specified language for program.

```
System.Globalization.CultureInfo Energy.Core.Program.SetLanguage(string culture)
```

Set default language for program (en-US).

```
System.Globalization.CultureInfo Energy.Core.Program.SetLanguage()
```

35.6 SetConsoleEncoding

Set specified encoding for console.

```
System.Text.Encoding Energy.Core.Program.SetConsoleEncoding(System.Text.Encoding␣
↪encoding)
```

```
System.Text.Encoding Energy.Core.Program.SetConsoleEncoding(string encoding)
```

Set encoding for console to UTF-8.

```
System.Text.Encoding Energy.Core.Program.SetConsoleEncoding()
```

CHAPTER 36

Query scripting

Helper functions for working with SQL query text.

36.1 Format

Value formatter class for SQL queries.

```
Energy.Query.Format format;
format = new Energy.Query.Format();
// or use global default which is obviously not recommended
format = Energy.Query.Format.Global;
```

36.1.1 Text

Format object value as TEXT.

Null values will be represented as “NULL”.

```
public string Text(string value)
```

Format object value as TEXT.

When nullify parameter is set to true, null values will be represented as “NULL” instead of “””.

```
string Text(string value, bool nullify)
```

Format object value as TEXT with limited length.

```
string Text(string text, int limit)
```

36.1.2 Unicode

Format as Unicode TEXT.

Null values will be represented as "NULL".

```
string Unicode(string value)
```

```
string Unicode(object value)
```

Format as Unicode TEXT.

When nullify parameter is set to true, null values will be represented as "NULL" instead of "".

```
string Unicode(string value, bool nullify)
```

```
string Unicode(object value, bool nullify)
```

36.1.3 Number

Format as NUMBER.

Real numbers are represented with dot "." as decimal point separator.

```
string Number(object value)
```

```
string Number(object value, bool nullify)
```

36.1.4 Integer

Format as INTEGER.

```
string Integer(int number)
```

Take only integer part of number.

```
string Integer(double number)
```

```
string Integer(decimal number)
```

Format as INTEGER.

Returns "1" for true and "0" for false.

```
string Integer(bool number)
```

36.1.5 Date

Format as DATE.

Represents date as quoted date string using "YYYY-MM-DD" format.

```
string Date(DateTime value)
```

```
string Date(object value)
```

36.1.6 Time

Format as TIME.

Uses 24h "hh:mm:ss" format.

Milliseconds will be used if present.

```
string Time(DateTime value)
```

```
string Time(object value)
```

36.1.7 Stamp

Format as DATETIME.

Uses by default "YYYY-MM-DD hh:mm:ss" format or "YYYY-MM-DD**T**hh:mm:ss" depending on settings.

```
string Stamp(DateTime value)
```

```
string Stamp(object value)
```

36.2 Type

36.2.1 Definition

Represents SQL database type definition from a string like "NVARCHAR(20) NOT NULL".

```
class Energy.Query.Type.Definition
{
    /// <summary>
    /// Represents type name.
    /// Example "VARCHAR".
    /// </summary>
    public string Type;

    /// <summary>
    /// Represents type parameter string.
    /// Example: "(9,2)".
    /// </summary>
    public string Parameter;

    /// <summary>
    /// Represents default option.
    /// Example: "DEFAULT ''"
    /// </summary>
    public string Default;
```

(continues on next page)

(continued from previous page)

```
    /// <summary>
    /// Represents nullable.
    /// Example "NOT NULL";
    /// </summary>
    public string Null;
}
```

36.2.2 Simplify

Simplify type.

36.2.3 IsNumeric

Check if SQL type is numeric.

```
bool Energy.Query.Type.IsNumeric(string type)
```

36.2.4 IsNullable

Check if SQL type is nullable.

```
bool Energy.Query.Type.IsNullable(string type)
```

36.3 Parameter

Support class for parametrized queries.

36.3.1 Bag

Represents list of parameters and their values. Use it to define parameters for parametrized query and to parse it.

```
Energy.Query.Parameter.Bag bag = new Energy.Query.Parameter.Bag();
bag.Set("param1", "value1");
bag.Set("param2", "value2");
string query = "INSERT INTO table1 ( column1 , column2 ) VALUES ( @param1 , @param2 )
↪";
string result = bag.Parse(query);
// expect "INSERT INTO table1 ( column1 , column2 ) VALUES ( 'value1' , 'value2' )";
```

This class also offers several options which may be set for proper parameter parsing process.

- Explicit

Parameters must be explicitly defined.

- NullAsZero

Parse null values as numeric zero.

- Unicode

Use N prefix for all non empty texts (Unicode).

- UnknownAsEmpty

Parse unknown parameters as empty texts. Does not apply to parameters with names with leading @@ (double at sign).

- UnknownAsNull

Parse unknown parameters as NULL. Does not apply to parameters with names with leading @@ (double at sign).

```
bag.Unicode = true;
bag.UnknownAsNull = true;
```

36.4 Template

Support for query templates.

36.4.1 ConvertToParameterizedQuery

Convert SQL query template which uses angle brackets to parameterized query which uses at sign to define parameters.

```
string template = @"
INSERT INTO [dbo].[Orders]
    ([number]
    ,[order date]
    )
VALUES
    (<number , nvarchar(35),>
    ,<order date , date,>
    )
GO
";
result = Energy.Query.Parameter.Template.ConvertToParameterizedQuery(template);
// expect "INSERT INTO [dbo].[Orders] ([number],[order date]) VALUES (@number,@order_
↪date)"
```

36.5 Quote

Quote string value using apostrophe (') as quotation mark. Function will return "NULL" if value is null.Exchange

```
string Energy.Query.Text.Quote(string value)
```

Quote string value using specified quotation mark. Use apostrophe (') for values and quotes (") for database object names. Function will return "NULL" if value is null.

```
string Energy.Query.Text.Quote(string value, char quote)
second)
```

Quote string value using specified quotation mark. Use apostrophe (') for values and quotes (") for database object names. Specify text to be returned when value is null or pass null to use default "NULL".

```
string Energy.Query.Text.Quote(string value, char quote, string nullText)
```

Quote string value using specified quotation mark. You might use square parenthesis ([]) to use T-SQL style quotation for database object names. Function will return "NULL" if value is null.

```
string Energy.Query.Text.Quote(string value, string quote)
```

Quote string value using specified quotation mark. You might use square parenthesis ([]) to use T-SQL style quotation for database object names. Specify text to be returned when value is null or pass null to use default "NULL".

```
string Energy.Query.Text.Quote(string value, string quote, string nullText)
```

36.6 Strip

Strip quotation from a value.

Return null if nullText parameter is specified and value equals to it.

```
string Energy.Query.Text.Strip(string value, string quote, string nullText)
```

Includes support for apostrophes, quotation marks, square brackets or unquoted values.

```
string Energy.Query.Text.Strip(string value, string quote, string nullText)
```

36.7 Definition

Represents SQL database type definition from a string like "NVARCHAR(20) NOT NULL".

```
Energy.Query.Definition def = Energy.Query.Type.ExtractTypeDefinition("VARCHAR(20)␣
↪NULL DEFAULT ''");
```

- Type

Represents type name. Example "VARCHAR".

- Simple

Represents simplified type name.

- Parameter

Represents type parameter string. Example: "(9,2)".

- Size

Size option.

36.8 ExtractTypeDefinition

Extract SQL data type for declaration string like "VARCHAR(50)"

```
Energy.Query.Definition def = Energy.Query.Type.ExtractTypeDefinition("VARCHAR(50)");
```

CHAPTER 37

Threading and workers

37.1 Worker

To be written…

37.2 Pool

To be written…

37.3 Example

To be written…

CHAPTER 38

Example: REST

This page contains REST client examples.

38.1 GET

Perform GET and return response from HTTP request.

```
string url = "https://www.google.com/search?q=Energy";
string body = Energy.Core.Web.Get(url).Body;
```

38.2 POST

Perform POST and return response from HTTP request.

```
string url = "http://localhost:12345/api/documents/";
string json = @"{ ""key1"": ""value1"" }";
string[] headers = new string[] { "X-Token", "1234567890" };
Energy.Base.Http.Response response;
response = Post(url, json, "application/json", headers);
```

38.3 PUT

Perform PUT and return response from URL.

```
Energy.Core.Web.IgnoreCertificateValidation = true;
string url = "https://reqres.in/api/users";
string body =
@"{
```

(continues on next page)

(continued from previous page)

```
    ""name"": ""morpheus"",
    ""job"": ""leader""
}";
Energy.Base.Http.Response response;
response = Energy.Core.Web.Put(url, body, "application/json");
```

38.4 Execute

More generic example.

```
var request = new Energy.Base.Http.Request("POST", methodUrl);
request.Encoding = System.Text.Encoding.UTF8;
request.Body = "{ \"text\": \"Gęś\" }";
request.ContentType = "application/javascript";
request.Headers.Add("X-Path: x-path");
var response = Energy.Core.Web.Execute(request);
```

38.5 Common problems

If POST or PUT doesn't work well, you might want to make sure you are specifying value for Content-Type like in the following example. While recommended way to perform web request is to use generic one (Request and Execute), you are able to set some headers using other methods.

```
string url = "https://reqres.in/api/users";
string body =
@"{
    ""name"": ""morpheus"",
    ""job"": ""leader""
}";
Energy.Base.Http.Response response;
Energy.Core.Web.IgnoreCertificateValidation = true;
response = Energy.Core.Web.Post(url, body, "application/json");
response = Energy.Core.Web.Patch(url, body, "application/json");
response = Energy.Core.Web.Put(url, body, "application/json");
```

CHAPTER 39

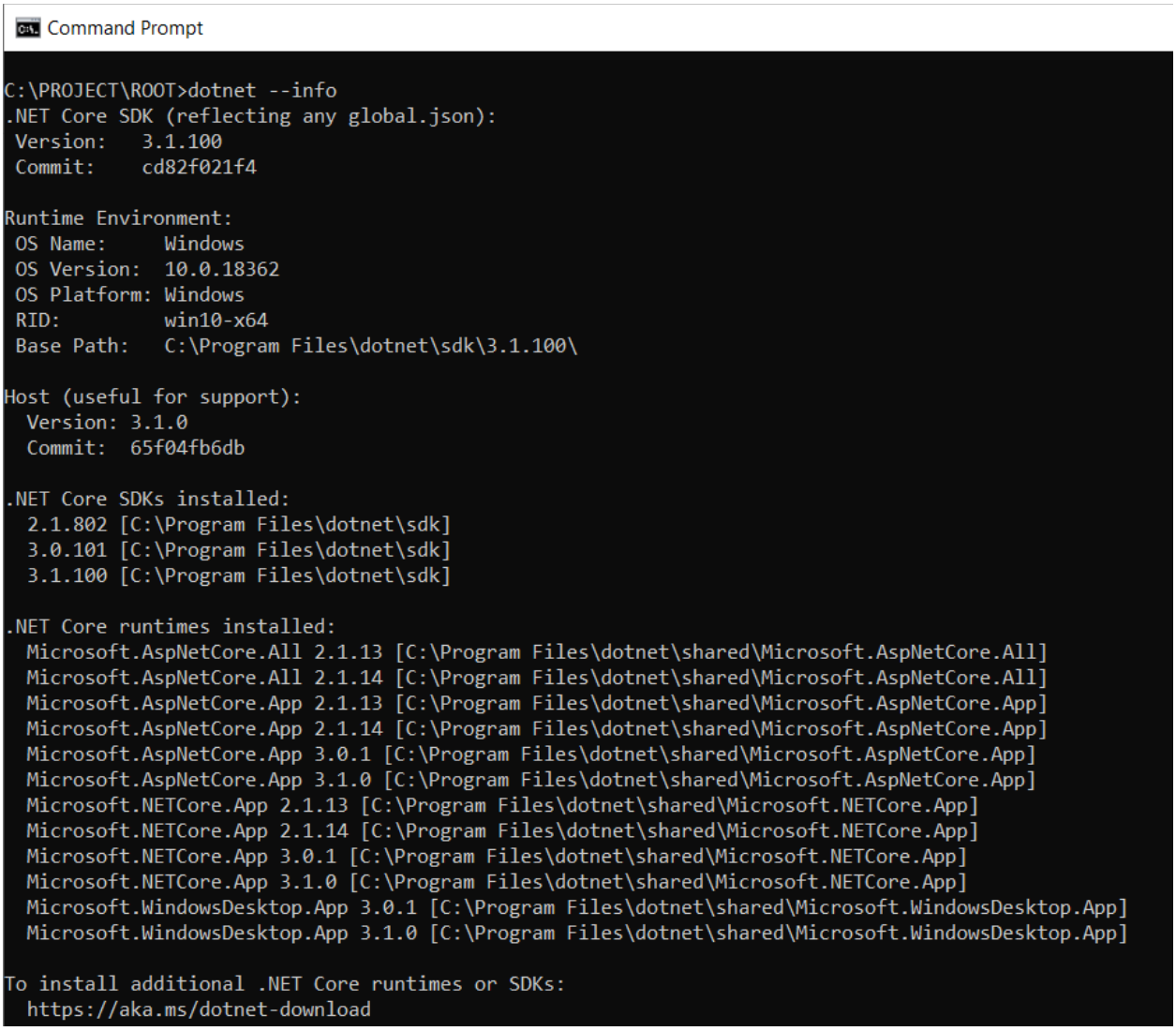
Example: WebApi

39.1 Basic example

Let't create new .NET project using dotnet and one of templates for web applications. Basic one.

```
dotnet new webapi --auth None -o AspNetCoreApi
```

If you do this for first time, take a look at SDK installed for .NET.




```
cd AspNetCoreApi
```

```
dotnet build
```

If you see “Build succeded”, project is ready to work on.

If you want to use just HTTP, you may comment out this line of code in your **Configure** method of your **Startup.cs**:

```
//app.UseHttpsRedirection();
```

39.1.1 Advanced example

Add package **Energy.Core** to your project.

```
dotnet add package Energy.Core
```

This example uses MySQL database, so additionally we need to install connector package.

```
dotnet add package MySql.Data
```

CHAPTER 40

Book

40.1 Code at first sight

Referencing “Energy.Core” library there

40.2 Case: Service

40.3 Case: Application

40.4 Borrow, play, throw away

Use it as you like.

40.5 Coding tips

40.5.1 Importing namespace

Warning about importing namespaces with “using” keyword. Consider always using full namespace or import using additional aliases for better future code maintenance in case of unpredictable changes like ESS.

```
using Energy.Enumeration as EE;
```

```
var textAlign = EE.TextAlign.Justify;
```

CHAPTER 41

Copyright

CHAPTER 42

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of this project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Credit for the logo goes to Damian Zaleski **zal3wa** at gmail.com.

CHAPTER 43

About

Energy.Core is a **.NET** class library which boosts your program with functionality covering various type conversions, utility classes, database abstraction layer, object mapping and simple application framework.

To be used by wide range of applications, services, web or console programs.

Filled with radioactive rays

Designed for all purposes as a set of different classes compiled into one library file.

It has minimal set of external dependencies and is available for several .NET platform versions.

Supports multithreading and asynchronous design patterns.

43.1 Compatibility

It's compatible with classic **.NET** plaftform versions **.NET 2.0**, **.NET 3.5**, **.NET 4.0** and later.

Supports **.NET Core** platform with minimum version **.NET Standard 2.0**.

.NET Compact Framework is also supported, available as separate download.

Some parts of library may use additional **.NET** libraries like *System.Xml.Serialization*.

CHAPTER 44

Download

The latest builds can be found at NuGet.

Package for Windows CE development needs to be downloaded directly from project page.