

LinksPlatform's Platform.Converters Class Library

./CachingConverterDecorator.cs

```
1 using System.Collections.Generic;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Converters
6 {
7     public class CachingConverterDecorator<TSource, TTarget> : IConverter<TSource, TTarget>
8     {
9         private readonly IConverter<TSource, TTarget> _baseConverter;
10        private readonly IDictionary<TSource, TTarget> _cache;
11
12        public CachingConverterDecorator(IConverter<TSource, TTarget> baseConverter,
13            ↪ IDictionary<TSource, TTarget> cache) => (_baseConverter, _cache) = (baseConverter,
14            ↪ cache);
15
16        public CachingConverterDecorator(IConverter<TSource, TTarget> baseConverter) :
17            ↪ this(baseConverter, new Dictionary<TSource, TTarget>()) { }
18
19        public TTarget Convert(TSource source)
20        {
21            if (!_cache.TryGetValue(source, out TTarget value))
22            {
23                value = _baseConverter.Convert(source);
24                _cache.Add(source, value);
25            }
26            return value;
27        }
28    }
29 }
```

./IConverter[T].cs

```
1 namespace Platform.Converters
2 {
3     /// <summary>
4     /// <para>Defines a converter between two values of the same type.</para>
5     /// <para>Определяет конвертер между двумя значениями одного типа.</para>
6     /// </summary>
7     /// <typeparam name="T"><para>Type of value to convert.</para><para>Тип преобразуемого
8     ↪ значения.</para></typeparam>
9     public interface IConverter<T> : IConverter<T, T>
10    {
11    }
12 }
```

./IConverter[TSource, TTarget].cs

```
1 namespace Platform.Converters
2 {
3     /// <summary>
4     /// <para>Defines a converter between two types (TSource and TTarget).</para>
5     /// <para>Определяет конвертер между двумя типами (исходным TSource и целевым
6     ↪ TTarget).</para>
7     /// </summary>
8     /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
9     ↪ конверсии.</para></typeparam>
10    /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
11    ↪ конверсии.</para></typeparam>
12    public interface IConverter<in TSource, out TTarget>
13    {
14        /// <summary>
15        /// <para>Converts the value of the source type (TSource) to the value of the target
16        ↪ type.</para>
17        /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
18        /// </summary>
19        /// <param name="source"><para>The source type value (TSource).</para><para>Значение
20        ↪ исходного типа (TSource).</para></param>
21        /// <returns><para>The value is converted to the target type
22        ↪ (TTarget).</para><para>Значение ковертированное в целевой тип
23        ↪ (TTarget).</para></returns>
24        TTarget Convert(TSource source);
25    }
26 }
```

./To.cs

```
1 using System;
2 using System.Runtime.CompilerServices;
```

```

3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Converters
7 {
8     public static class To
9     {
10         public static readonly char UnknownCharacter = '\0';
11
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public static ulong UInt64(ulong value) => value;
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public static long Int64(ulong value) => unchecked(value > long.MaxValue ? long.MaxValue
17             ↪ : (long)value);
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public static uint UInt32(ulong value) => unchecked(value > uint.MaxValue ?
21             ↪ uint.MaxValue : (uint)value);
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static int Int32(ulong value) => unchecked(value > int.MaxValue ? int.MaxValue :
25             ↪ (int)value);
26
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public static ushort UInt16(ulong value) => unchecked(value > ushort.MaxValue ?
29             ↪ ushort.MaxValue : (ushort)value);
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public static short Int16(ulong value) => unchecked(value > (ulong)short.MaxValue ?
33             ↪ short.MaxValue : (short)value);
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public static byte Byte(ulong value) => unchecked(value > byte.MaxValue ? byte.MaxValue
37             ↪ : (byte)value);
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public static sbyte SByte(ulong value) => unchecked(value > (ulong)sbyte.MaxValue ?
41             ↪ sbyte.MaxValue : (sbyte)value);
42
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public static bool Boolean(ulong value) => value > 0UL;
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static char Char(ulong value) => unchecked(value > char.MaxValue ?
48             ↪ UnknownCharacter : (char)value);
49
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public static DateTime DateTime(ulong value) => unchecked(value > long.MaxValue ?
52             ↪ System.DateTime.MaxValue : new DateTime((long)value));
53
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public static TimeSpan TimeSpan(ulong value) => unchecked(value > long.MaxValue ?
56             ↪ System.TimeSpan.MaxValue : new TimeSpan((long)value));
57
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public static ulong UInt64(long value) => unchecked(value < (long)ulong.MinValue ?
60             ↪ ulong.MinValue : (ulong)value);
61
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public static ulong UInt64(int value) => unchecked(value < (int)ulong.MinValue ?
64             ↪ ulong.MinValue : (ulong)value);
65
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         public static ulong UInt64(short value) => unchecked(value < (short)ulong.MinValue ?
68             ↪ ulong.MinValue : (ulong)value);
69
70         [MethodImpl(MethodImplOptions.AggressiveInlining)]
71         public static ulong UInt64(sbyte value) => unchecked(value < (sbyte)ulong.MinValue ?
72             ↪ ulong.MinValue : (ulong)value);
73
74         [MethodImpl(MethodImplOptions.AggressiveInlining)]
75         public static ulong UInt64(bool value) => value ? 1UL : 0UL;
76
77         [MethodImpl(MethodImplOptions.AggressiveInlining)]
78         public static ulong UInt64(char value) => value;
79
80         [MethodImpl(MethodImplOptions.AggressiveInlining)]
81

```

```

67     public static long Signed(ulong value) => unchecked((long)value);
68
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     public static int Signed(uint value) => unchecked((int)value);
71
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     public static short Signed(ushort value) => unchecked((short)value);
74
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     public static sbyte Signed(byte value) => unchecked((sbyte)value);
77
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     public static object Signed<T>(T value) => To<T>.Signed(value);
80
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     public static ulong Unsigned(long value) => unchecked((ulong)value);
83
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     public static uint Unsigned(int value) => unchecked((uint)value);
86
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     public static ushort Unsigned(short value) => unchecked((ushort)value);
89
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public static byte Unsigned(sbyte value) => unchecked((byte)value);
92
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     public static object Unsigned<T>(T value) => To<T>.Unsigned(value);
95
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public static T UnsignedAs<T>(object value) => To<T>.UnsignedAs(value);
98 }
99 }

```

./To[T].cs

```

1  using System;
2  using Platform.Exceptions;
3  using Platform.Reflection;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Converters
8  {
9      public static class To<T>
10     {
11         public static readonly Func<T, object> Signed;
12         public static readonly Func<T, object> Unsigned;
13         public static readonly Func<object, T> UnsignedAs;
14
15         static To()
16         {
17             Signed = CompileSignedDelegate();
18             Unsigned = CompileUnsignedDelegate();
19             UnsignedAs = CompileUnsignedAsDelegate();
20         }
21
22         static private Func<T, object> CompileSignedDelegate()
23         {
24             return DelegateHelpers.Compile<Func<T, object>>(emitter =>
25             {
26                 Ensure.Always.IsUnsignedInteger<T>();
27                 emitter.LoadArgument(0);
28                 var method = typeof(To).GetMethod("Signed", Types<T>.Array);
29                 emitter.Call(method);
30                 emitter.Box(method.ReturnType);
31                 emitter.Return();
32             });
33         }
34
35         static private Func<T, object> CompileUnsignedDelegate()
36         {
37             return DelegateHelpers.Compile<Func<T, object>>(emitter =>
38             {
39                 Ensure.Always.IsSignedInteger<T>();
40                 emitter.LoadArgument(0);
41                 var method = typeof(To).GetMethod("Unsigned", Types<T>.Array);
42                 emitter.Call(method);
43                 emitter.Box(method.ReturnType);
44                 emitter.Return();
45             });
46         }
47     }
48 }

```

```
46     }
47
48     static private Func<object, T> CompileUnsignedAsDelegate()
49     {
50         return DelegateHelpers.Compile<Func<object, T>>(emitter =>
51         {
52             Ensure.Always.IsUnsignedInteger<T>();
53             emitter.LoadArgument(0);
54             var signedVersion = NumericType<T>.SignedVersion;
55             emitter.UnboxValue(signedVersion);
56             var method = typeof(To).GetMethod("Unsigned", new[] { signedVersion });
57             emitter.Call(method);
58             emitter.Return();
59         });
60     }
61 }
62 }
```

Index

./CachingConverterDecorator.cs, 1
./IConverter[TSource, TTarget].cs, 1
./IConverter[T].cs, 1
./To.cs, 1
./To[T].cs, 3