

LinksPlatform's Platform.Converters Class Library

1.1 ./Platform.Converters/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 }
```

1.2 ./Platform.Converters/CheckedConverter.cs

```
1 using System;
2 using System.Reflection;
3 using System.Reflection.Emit;
4 using Platform.Reflection;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Converters
9 {
10    public abstract class CheckedConverter<TSource, TTarget> : IConverter<TSource, TTarget>
11    {
12        public static CheckedConverter<TSource, TTarget> Default { get; }
13
14        static CheckedConverter()
15        {
16            AssemblyName assemblyName = new AssemblyName(GetNewName());
17            var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,
18                ↪ AssemblyBuilderAccess.Run);
19            var module = assembly.DefineDynamicModule(GetNewName());
20            var type = module.DefineType(GetNewName(), TypeAttributes.Public |
21                ↪ TypeAttributes.Class | TypeAttributes.Sealed, typeof(CheckedConverter<TSource,
22                ↪ TTarget>));
23            type.EmitVirtualMethod<Converter<TSource, TTarget>>("Convert", il =>
24            {
25                il.LoadArgument(1);
26                if (typeof(TSource) != typeof(TTarget))
27                {
28                    il.CheckedConvert<TSource, TTarget>();
29                }
30                il.Return();
31            });
32            var typeInfo = type.CreateTypeInfo();
33            Default = (CheckedConverter<TSource, TTarget>)Activator.CreateInstance(typeInfo);
34        }
35
36        private static string GetNewName() => Guid.NewGuid().ToString("N");
37
38        public abstract TTarget Convert(TSource source);
39    }
40 }
```

1.3 ./Platform.Converters/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
   ↳ TTarget).</para>
6  /// </summary>
7  /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
   ↳ конверсии.</para></typeparam>
8  /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
   ↳ конверсии.</para></typeparam>
9  public interface IConverter<in TSource, out TTarget>
10 {
11     /// <summary>
12     /// <para>Converts the value of the source type (TSource) to the value of the target
   ↳ type.</para>
13     /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
14     /// </summary>
15     /// <param name="source"><para>The source type value (TSource).</para><para>Значение
   ↳ исходного типа (TSource).</para></param>
16     /// <returns><para>The value is converted to the target type
   ↳ (TTarget).</para><para>Значение конвертированное в целевой тип
   ↳ (TTarget).</para></returns>
17     TTarget Convert(TSource source);
18 }
19 }

```

1.4 ./Platform.Converters/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
   ↳ значения.</para></typeparam>
8      public interface IConverter<T> : IConverter<T, T>
9      {
10     }
11 }

```

1.5 ./Platform.Converters/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      [Obsolete]
9      public static class To
10     {
11         public static readonly char UnknownCharacter = '\0';
12
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public static ulong UInt64(ulong value) => value;
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         public static long Int64(ulong value) => unchecked(value > long.MaxValue ? long.MaxValue
   ↳ : (long)value);
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public static uint UInt32(ulong value) => unchecked(value > uint.MaxValue ?
   ↳ uint.MaxValue : (uint)value);
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         public static int Int32(ulong value) => unchecked(value > int.MaxValue ? int.MaxValue :
   ↳ (int)value);
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         public static ushort UInt16(ulong value) => unchecked(value > ushort.MaxValue ?
   ↳ ushort.MaxValue : (ushort)value);
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public static short Int16(ulong value) => unchecked(value > (ulong)short.MaxValue ?
   ↳ short.MaxValue : (short)value);
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public static byte Byte(ulong value) => unchecked(value > byte.MaxValue ? byte.MaxValue
   ↳ : (byte)value);
33     }

```

```

34 [MethodImpl(MethodImplOptions.AggressiveInlining)]
35 public static sbyte SByte(ulong value) => unchecked(value > (ulong)sbyte.MaxValue ?
    ↳ sbyte.MaxValue : (sbyte)value);
36
37 [MethodImpl(MethodImplOptions.AggressiveInlining)]
38 public static bool Boolean(ulong value) => value > OUL;
39
40 [MethodImpl(MethodImplOptions.AggressiveInlining)]
41 public static char Char(ulong value) => unchecked(value > char.MaxValue ?
    ↳ UnknownCharacter : (char)value);
42
43 [MethodImpl(MethodImplOptions.AggressiveInlining)]
44 public static DateTime DateTime(ulong value) => unchecked(value > long.MaxValue ?
    ↳ System.DateTime.MaxValue : new DateTime((long)value));
45
46 [MethodImpl(MethodImplOptions.AggressiveInlining)]
47 public static TimeSpan TimeSpan(ulong value) => unchecked(value > long.MaxValue ?
    ↳ System.TimeSpan.MaxValue : new TimeSpan((long)value));
48
49 [MethodImpl(MethodImplOptions.AggressiveInlining)]
50 public static ulong UInt64(long value) => unchecked(value < (long)ulong.MinValue ?
    ↳ ulong.MinValue : (ulong)value);
51
52 [MethodImpl(MethodImplOptions.AggressiveInlining)]
53 public static ulong UInt64(int value) => unchecked(value < (int)ulong.MinValue ?
    ↳ ulong.MinValue : (ulong)value);
54
55 [MethodImpl(MethodImplOptions.AggressiveInlining)]
56 public static ulong UInt64(short value) => unchecked(value < (short)ulong.MinValue ?
    ↳ ulong.MinValue : (ulong)value);
57
58 [MethodImpl(MethodImplOptions.AggressiveInlining)]
59 public static ulong UInt64(sbyte value) => unchecked(value < (sbyte)ulong.MinValue ?
    ↳ ulong.MinValue : (ulong)value);
60
61 [MethodImpl(MethodImplOptions.AggressiveInlining)]
62 public static ulong UInt64(bool value) => value ? 1UL : OUL;
63
64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 public static ulong UInt64(char value) => value;
66
67 [MethodImpl(MethodImplOptions.AggressiveInlining)]
68 public static long Signed(ulong value) => unchecked((long)value);
69
70 [MethodImpl(MethodImplOptions.AggressiveInlining)]
71 public static int Signed(uint value) => unchecked((int)value);
72
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 public static short Signed(ushort value) => unchecked((short)value);
75
76 [MethodImpl(MethodImplOptions.AggressiveInlining)]
77 public static sbyte Signed(byte value) => unchecked((sbyte)value);
78
79 [MethodImpl(MethodImplOptions.AggressiveInlining)]
80 public static object Signed<T>(T value) => To<T>.Signed(value);
81
82 [MethodImpl(MethodImplOptions.AggressiveInlining)]
83 public static ulong Unsigned(long value) => unchecked((ulong)value);
84
85 [MethodImpl(MethodImplOptions.AggressiveInlining)]
86 public static uint Unsigned(int value) => unchecked((uint)value);
87
88 [MethodImpl(MethodImplOptions.AggressiveInlining)]
89 public static ushort Unsigned(short value) => unchecked((ushort)value);
90
91 [MethodImpl(MethodImplOptions.AggressiveInlining)]
92 public static byte Unsigned(sbyte value) => unchecked((byte)value);
93
94 [MethodImpl(MethodImplOptions.AggressiveInlining)]
95 public static object Unsigned<T>(T value) => To<T>.Unsigned(value);
96
97 [MethodImpl(MethodImplOptions.AggressiveInlining)]
98 public static T UnsignedAs<T>(object value) => To<T>.UnsignedAs(value);
99 }
100 }

```

1.6 ./Platform.Converters/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     [Obsolete]
10    public static class To<T>
11    {
12        public static readonly Func<T, object> Signed = CompileSignedDelegate();
13        public static readonly Func<T, object> Unsigned = CompileUnsignedDelegate();
14        public static readonly Func<object, T> UnsignedAs = CompileUnsignedAsDelegate();
15
16        static private Func<T, object> CompileSignedDelegate()
17        {
18            return DelegateHelpers.Compile<Func<T, object>>(emitter =>
19            {
20                Ensure.Always.IsUnsignedInteger<T>();
21                emitter.LoadArgument(0);
22                var method = typeof(To).GetMethod("Signed", Types<T>.Array);
23                emitter.Call(method);
24                emitter.Box(method.ReturnType);
25                emitter.Return();
26            });
27        }
28
29        static private Func<T, object> CompileUnsignedDelegate()
30        {
31            return DelegateHelpers.Compile<Func<T, object>>(emitter =>
32            {
33                Ensure.Always.IsSignedInteger<T>();
34                emitter.LoadArgument(0);
35                var method = typeof(To).GetMethod("Unsigned", Types<T>.Array);
36                emitter.Call(method);
37                emitter.Box(method.ReturnType);
38                emitter.Return();
39            });
40        }
41
42        static private Func<object, T> CompileUnsignedAsDelegate()
43        {
44            return DelegateHelpers.Compile<Func<object, T>>(emitter =>
45            {
46                Ensure.Always.IsUnsignedInteger<T>();
47                emitter.LoadArgument(0);
48                var signedVersion = NumericType<T>.SignedVersion;
49                emitter.UnboxValue(signedVersion);
50                var method = typeof(To).GetMethod("Unsigned", new[] { signedVersion });
51                emitter.Call(method);
52                emitter.Return();
53            });
54        }
55    }
56 }

```

1.7 ./Platform.Converters/UncheckedConverter.cs

```

1 using System;
2 using System.Reflection;
3 using System.Reflection.Emit;
4 using System.Runtime.CompilerServices;
5 using Platform.Reflection;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Converters
10 {
11     public abstract class UncheckedConverter<TSource, TTarget> : IConverter<TSource, TTarget>
12     {
13         public static UncheckedConverter<TSource, TTarget> Default { get; }
14
15         static UncheckedConverter()
16         {
17             AssemblyName assemblyName = new AssemblyName(GetNewName());
18             var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,
19                 ↳ AssemblyBuilderAccess.Run);
20             var module = assembly.DefineDynamicModule(GetNewName());
21             var type = module.DefineType(GetNewName(), TypeAttributes.Public |
22                 ↳ TypeAttributes.Class | TypeAttributes.Sealed, typeof(UncheckedConverter<TSource,
23                 ↳ TTarget>));
24         }
25     }
26 }

```

```

21         type.EmitVirtualMethod<Converter<TSource, TTarget>>("Convert", il =>
22         {
23             il.LoadArgument(1);
24             if (typeof(TSource) != typeof(TTarget))
25             {
26                 il.UncheckedConvert<TSource, TTarget>();
27             }
28             il.Return();
29         });
30         var typeInfo = type.CreateTypeInfo();
31         Default = (UncheckedConverter<TSource, TTarget>)Activator.CreateInstance(typeInfo);
32     }
33
34     private static string GetNewName() => Guid.NewGuid().ToString("N");
35
36     [MethodImpl(MethodImplOptions.AggressiveInlining)]
37     public abstract TTarget Convert(TSource source);
38 }
39 }

```

1.8 ./Platform.Converters.Tests/ConverterTests.cs

```

1  using Platform.Diagnostics;
2  using System;
3  using System.Globalization;
4  using System.Runtime.CompilerServices;
5  using Xunit;
6  using Xunit.Abstractions;
7
8  namespace Platform.Converters.Tests
9  {
10     public class ConverterTests
11     {
12         private readonly ITestOutputHelper _output;
13         private static readonly UncheckedConverter<ulong, ulong> _uInt64ToUInt64Converter =
14             ↪ UncheckedConverter<ulong, ulong>.Default;
15         private static readonly UncheckedConverter<int, ulong> _int32ToUInt64converter =
16             ↪ UncheckedConverter<int, ulong>.Default;
17
18         public ConverterTests(ITestOutputHelper output) => _output = output;
19
20         [Fact]
21         public void SameTypeTest()
22         {
23             var result = UncheckedConverter<ulong, ulong>.Default.Convert(2UL);
24             Assert.Equal(2UL, result);
25             result = CheckedConverter<ulong, ulong>.Default.Convert(2UL);
26             Assert.Equal(2UL, result);
27         }
28
29         [Fact]
30         public void SameTypePerformanceComparisonTest()
31         {
32             var N = 10000000;
33             var result = 0UL;
34
35             // Warmup
36             for (int i = 0; i < N; i++)
37             {
38                 result = _uInt64ToUInt64Converter.Convert(2UL);
39             }
40             for (int i = 0; i < N; i++)
41             {
42                 result = UncheckedConverter<ulong, ulong>.Default.Convert(2UL);
43             }
44             for (int i = 0; i < N; i++)
45             {
46                 result = Convert(2UL);
47             }
48             for (int i = 0; i < N; i++)
49             {
50                 result = To.UInt64(2UL);
51             }
52             for (int i = 0; i < N; i++)
53             {
54                 result = System.Convert.ToUInt64(2UL);
55             }
56             for (int i = 0; i < N; i++)
57             {
58                 result = (ulong)System.Convert.ChangeType(2UL, typeof(ulong));
59             }
60         }
61     }
62 }

```

```

57     }
58
59     var ts1 = Performance.Measure(() =>
60     {
61         for (int i = 0; i < N; i++)
62         {
63             result = _uInt64ToUInt64Converter.Convert(2UL);
64         }
65     });
66     var ts2 = Performance.Measure(() =>
67     {
68         for (int i = 0; i < N; i++)
69         {
70             result = UncheckedConverter<ulong, ulong>.Default.Convert(2UL);
71         }
72     });
73     var ts3 = Performance.Measure(() =>
74     {
75         for (int i = 0; i < N; i++)
76         {
77             result = Convert(2UL);
78         }
79     });
80     var ts4 = Performance.Measure(() =>
81     {
82         for (int i = 0; i < N; i++)
83         {
84             result = To.UInt64(2UL);
85         }
86     });
87     var ts5 = Performance.Measure(() =>
88     {
89         for (int i = 0; i < N; i++)
90         {
91             result = System.Convert.ToUInt64(2UL);
92         }
93     });
94     var ts6 = Performance.Measure(() =>
95     {
96         for (int i = 0; i < N; i++)
97         {
98             result = (ulong)System.Convert.ChangeType(2UL, typeof(ulong));
99         }
100    });
101    IFormatProvider formatProvider = CultureInfo.InvariantCulture;
102    var ts7 = Performance.Measure(() =>
103    {
104        for (int i = 0; i < N; i++)
105        {
106            result = ((IConvertible)2UL).ToUInt64(formatProvider);
107        }
108    });
109    var ts8 = Performance.Measure(() =>
110    {
111        for (int i = 0; i < N; i++)
112        {
113            result = (ulong)((IConvertible)2UL).ToType(typeof(ulong), formatProvider);
114        }
115    });
116
117    _output.WriteLine($"{ts1} {ts2} {ts3} {ts4} {ts5} {ts6} {ts7} {ts8} {result}");
118 }
119
120 [Fact]
121 public void Int32ToUInt64Test()
122 {
123     var result = UncheckedConverter<int, ulong>.Default.Convert(2);
124     Assert.Equal(2UL, result);
125     result = CheckedConverter<int, ulong>.Default.Convert(2);
126     Assert.Equal(2UL, result);
127 }
128
129 [Fact]
130 public void Int32ToUInt64PerformanceComparisonTest()
131 {
132     var N = 10000000;
133     var result = 0UL;
134

```

```

135 // Warmup
136 for (int i = 0; i < N; i++)
137 {
138     result = _int32ToUInt64converter.Convert(2);
139 }
140 for (int i = 0; i < N; i++)
141 {
142     result = UncheckedConverter<ulong, ulong>.Default.Convert(2);
143 }
144 for (int i = 0; i < N; i++)
145 {
146     result = Convert(2);
147 }
148 for (int i = 0; i < N; i++)
149 {
150     result = To.UInt64(2);
151 }
152 for (int i = 0; i < N; i++)
153 {
154     result = System.Convert.ToUInt64(2);
155 }
156 for (int i = 0; i < N; i++)
157 {
158     result = (ulong)System.Convert.ChangeType(2, typeof(ulong));
159 }
160
161 var ts1 = Performance.Measure(() =>
162 {
163     for (int i = 0; i < N; i++)
164     {
165         result = _int32ToUInt64converter.Convert(2);
166     }
167 });
168 var ts2 = Performance.Measure(() =>
169 {
170     for (int i = 0; i < N; i++)
171     {
172         result = UncheckedConverter<ulong, ulong>.Default.Convert(2);
173     }
174 });
175 var ts3 = Performance.Measure(() =>
176 {
177     for (int i = 0; i < N; i++)
178     {
179         result = Convert(2);
180     }
181 });
182 var ts4 = Performance.Measure(() =>
183 {
184     for (int i = 0; i < N; i++)
185     {
186         result = To.UInt64(2);
187     }
188 });
189 var ts5 = Performance.Measure(() =>
190 {
191     for (int i = 0; i < N; i++)
192     {
193         result = System.Convert.ToUInt64(2);
194     }
195 });
196 var ts6 = Performance.Measure(() =>
197 {
198     for (int i = 0; i < N; i++)
199     {
200         result = (ulong)System.Convert.ChangeType(2, typeof(ulong));
201     }
202 });
203 IFormatProvider formatProvider = CultureInfo.InvariantCulture;
204 var ts7 = Performance.Measure(() =>
205 {
206     for (int i = 0; i < N; i++)
207     {
208         result = ((IConvertible)2).ToUInt64(formatProvider);
209     }
210 });
211 var ts8 = Performance.Measure(() =>
212 {

```

```
213         for (int i = 0; i < N; i++)
214         {
215             result = (ulong)((IConvertible)2).ToType(typeof(ulong), formatProvider);
216         }
217     });
218
219     _output.WriteLine($"{ts1} {ts2} {ts3} {ts4} {ts5} {ts6} {ts7} {ts8} {result}");
220 }
221
222 [MethodImpl(MethodImplOptions.AggressiveInlining)]
223 public static ulong Convert(ulong value) => _uInt64ToUInt64Converter.Convert(value);
224
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 public static ulong Convert(int value) => _int32ToUInt64converter.Convert(value);
227 }
228 }
```


Index

- ./Platform.Converters.Tests/ConverterTests.cs, 5
- ./Platform.Converters/CachingConverterDecorator.cs, 1
- ./Platform.Converters/CheckedConverter.cs, 1
- ./Platform.Converters/IConverter[TSource, TTarget].cs, 1
- ./Platform.Converters/IConverter[T].cs, 2
- ./Platform.Converters/To.cs, 2
- ./Platform.Converters/To[T].cs, 3
- ./Platform.Converters/UncheckedConverter.cs, 4