

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.Linq;
3 using System.Reflection;
4 using System.Reflection.Emit;
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 CheckedConverter<TSource, TTarget> : IConverter<TSource, TTarget>
12     {
13         public static CheckedConverter<TSource, TTarget> Default { get; }
14
15         static CheckedConverter()
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(CheckedConverter<TSource,
23                 ↪ TTarget>));
24             EmitMethod<Converter<TSource, TTarget>>(type, "Convert", (il) =>
25             {
26                 il.LoadArgument(1);
27                 if (typeof(TSource) != typeof(TTarget))
28                 {
29                     CheckedConvert(il);
30                 }
31                 il.Return();
32             });
33             var typeInfo = type.CreateTypeInfo();
34             Default = (CheckedConverter<TSource, TTarget>)Activator.CreateInstance(typeInfo);
35         }
36
37         private static void CheckedConvert(ILGenerator generator)
38         {
39             var type = typeof(TTarget);
40             if (type == typeof(short))
41             {
42                 if (NumericType<TSource>.IsSigned)
43                 {
44                     generator.Emit(OpCodes.Conv_Ovf_I2);
45                 }
46                 else
47                 {
48                     generator.Emit(OpCodes.Conv_U2);
49                 }
50             }
51             else
52             {
53                 generator.Emit(OpCodes.Castclass, type);
54             }
55         }
56     }
57 }
```

```

44         {
45             generator.Emit(OpCodes.Conv_Ovf_I2_Un);
46         }
47     }
48     else if (type == typeof(ushort))
49     {
50         if (NumericType<TSource>.IsSigned)
51         {
52             generator.Emit(OpCodes.Conv_Ovf_U2);
53         }
54         else
55         {
56             generator.Emit(OpCodes.Conv_Ovf_U2_Un);
57         }
58     }
59     else if (type == typeof(sbyte))
60     {
61         if (NumericType<TSource>.IsSigned)
62         {
63             generator.Emit(OpCodes.Conv_Ovf_I1);
64         }
65         else
66         {
67             generator.Emit(OpCodes.Conv_Ovf_I1_Un);
68         }
69     }
70     else if (type == typeof(byte))
71     {
72         if (NumericType<TSource>.IsSigned)
73         {
74             generator.Emit(OpCodes.Conv_Ovf_U1);
75         }
76         else
77         {
78             generator.Emit(OpCodes.Conv_Ovf_U1_Un);
79         }
80     }
81     else if (type == typeof(int))
82     {
83         if (NumericType<TSource>.IsSigned)
84         {
85             generator.Emit(OpCodes.Conv_Ovf_I4);
86         }
87         else
88         {
89             generator.Emit(OpCodes.Conv_Ovf_I4_Un);
90         }
91     }
92     else if (type == typeof(uint))
93     {
94         if (NumericType<TSource>.IsSigned)
95         {
96             generator.Emit(OpCodes.Conv_Ovf_U4);
97         }
98         else
99         {
100             generator.Emit(OpCodes.Conv_Ovf_U4_Un);
101         }
102     }
103     else if (type == typeof(long))
104     {
105         if (NumericType<TSource>.IsSigned)
106         {
107             generator.Emit(OpCodes.Conv_Ovf_I8);
108         }
109         else
110         {
111             generator.Emit(OpCodes.Conv_Ovf_I8_Un);
112         }
113     }
114     else if (type == typeof(ulong))
115     {
116         if (NumericType<TSource>.IsSigned)
117         {
118             generator.Emit(OpCodes.Conv_Ovf_U8);
119         }
120         else
121         {

```

```

122         generator.Emit(OpCodes.Conv_Ovf_U8_Un);
123     }
124 }
125 else if (type == typeof(float))
126 {
127     if (NumericType<TSource>.IsSigned)
128     {
129         generator.Emit(OpCodes.Conv_R4);
130     }
131     else
132     {
133         generator.Emit(OpCodes.Conv_R_Un);
134     }
135 }
136 else if (type == typeof(double))
137 {
138     generator.Emit(OpCodes.Conv_R8);
139 }
140 else
141 {
142     throw new NotSupportedException();
143 }
144 }
145
146 private static void EmitMethod<TDelegate>(TypeBuilder type, string methodName,
147     ↪ Action<ILGenerator> emitCode)
148 {
149     var delegateType = typeof(TDelegate);
150     var invoke = delegateType.GetMethod("Invoke");
151     var returnType = invoke.ReturnType;
152     var parameterTypes = invoke.GetParameters().Select(s => s.ParameterType).ToArray();
153     MethodBuilder method = type.DefineMethod(methodName, MethodAttributes.Public |
154     ↪ MethodAttributes.Virtual | MethodAttributes.Final | MethodAttributes.HideBySig,
155     ↪ returnType, parameterTypes);
156     method.SetImplementationFlags(MethodImplAttributes.IL | MethodImplAttributes.Managed
157     ↪ | MethodImplAttributes.AggressiveInlining);
158     var generator = method.GetILGenerator();
159     emitCode(generator);
160 }
161
162 private static string GetNewName() => Guid.NewGuid().ToString("N");
163
164 public abstract TTarget Convert(TSource source);
165 }
166 }

```

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
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 конвертированное в целевой тип
23        ↪ (TTarget).</para></returns>
24        TTarget Convert(TSource source);
25    }
26 }

```

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 > 0UL;
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 : 0UL;
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.Linq;
3  using System.Reflection;
4  using System.Reflection.Emit;
5  using System.Runtime.CompilerServices;
6  using Platform.Reflection;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Converters
11 {
12     public abstract class UncheckedConverter<TSource, TTarget> : IConverter<TSource, TTarget>
13     {
14         public static UncheckedConverter<TSource, TTarget> Default { get; }
15
16         static UncheckedConverter()
17         {
18             AssemblyName assemblyName = new AssemblyName(GetNewName());
19             var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,
20                 ↳ AssemblyBuilderAccess.Run);
21             var module = assembly.DefineDynamicModule(GetNewName());
22             var type = module.DefineType(GetNewName(), TypeAttributes.Public |
23                 ↳ TypeAttributes.Class | TypeAttributes.Sealed, typeof(UncheckedConverter<TSource,
24                 ↳ TTarget>));
25             EmitMethod<Converter<TSource, TTarget>>(type, "Convert", (il) =>
26             {
27                 il.LoadArgument(1);
28                 if (typeof(TSource) != typeof(TTarget))
29                 {
30                     UncheckedConvert(il);
31                 }
32                 il.Return();
33             });
34             var typeInfo = type.CreateTypeInfo();
35             Default = (UncheckedConverter<TSource, TTarget>)Activator.CreateInstance(typeInfo);
36         }
37
38         private static void UncheckedConvert(ILGenerator generator)
39         {
40             var type = typeof(TTarget);
41             if (type == typeof(short))
42             {
43                 generator.Emit(OpCodes.Conv_I2);
44             }
45             else if (type == typeof(ushort))
46             {
47                 generator.Emit(OpCodes.Conv_U2);
48             }
49             else if (type == typeof(sbyte))
50             {
51

```

```

48         generator.Emit(OpCodes.Conv_I1);
49     }
50     else if (type == typeof(byte))
51     {
52         generator.Emit(OpCodes.Conv_U1);
53     }
54     else if (type == typeof(int))
55     {
56         generator.Emit(OpCodes.Conv_I4);
57     }
58     else if (type == typeof(uint))
59     {
60         generator.Emit(OpCodes.Conv_U4);
61     }
62     else if (type == typeof(long))
63     {
64         generator.Emit(OpCodes.Conv_I8);
65     }
66     else if (type == typeof(ulong))
67     {
68         generator.Emit(OpCodes.Conv_U8);
69     }
70     else if (type == typeof(float))
71     {
72         if (NumericType<TSource>.IsSigned)
73         {
74             generator.Emit(OpCodes.Conv_R4);
75         }
76         else
77         {
78             generator.Emit(OpCodes.Conv_R_Un);
79         }
80     }
81     else if (type == typeof(double))
82     {
83         generator.Emit(OpCodes.Conv_R8);
84     }
85     else
86     {
87         throw new NotSupportedException();
88     }
89 }
90
91 private static void EmitMethod<TDelegate>(TypeBuilder type, string methodName,
92     ↪ Action<ILGenerator> emitCode)
93 {
94     var delegateType = typeof(TDelegate);
95     var invoke = delegateType.GetMethod("Invoke");
96     var returnType = invoke.ReturnType;
97     var parameterTypes = invoke.GetParameters().Select(s => s.ParameterType).ToArray();
98     MethodBuilder method = type.DefineMethod(methodName, MethodAttributes.Public |
99     ↪ MethodAttributes.Virtual | MethodAttributes.Final | MethodAttributes.HideBySig,
100     ↪ returnType, parameterTypes);
101     method.SetImplementationFlags(MethodImplAttributes.IL | MethodImplAttributes.Managed
102     ↪ | MethodImplAttributes.AggressiveInlining);
103     var generator = method.GetILGenerator();
104     emitCode(generator);
105 }
106
107 private static string GetNewName() => Guid.NewGuid().ToString("N");
108
109 [MethodImpl(MethodImplOptions.AggressiveInlining)]
110 public abstract TTarget Convert(TSource source);
111 }
112 }

```

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     var ts1 = Performance.Measure(() =>
62     {
63         for (int i = 0; i < N; i++)
64         {
65             result = _uInt64ToUInt64Converter.Convert(2UL);
66         }
67     });
68     var ts2 = Performance.Measure(() =>
69     {
70         for (int i = 0; i < N; i++)
71         {
72             result = UncheckedConverter<ulong, ulong>.Default.Convert(2UL);
73         }
74     });
75     var ts3 = Performance.Measure(() =>
76     {
77         for (int i = 0; i < N; i++)
78         {
79             result = Convert(2UL);
80         }
81     });
82     var ts4 = Performance.Measure(() =>
83     {
84         for (int i = 0; i < N; i++)
85         {
86             result = To.UInt64(2UL);
87         }
88     });
89     var ts5 = Performance.Measure(() =>
90     {
91         for (int i = 0; i < N; i++)
92         {
93             result = System.Convert.ToUInt64(2UL);
94         }
95     });
96     var ts6 = Performance.Measure(() =>
97     {
98         for (int i = 0; i < N; i++)
99         {
100             result = (ulong)System.Convert.ChangeType(2UL, typeof(ulong));
101         }
102     });
103
104     Console.WriteLine($"SameTypeTest: ts1={ts1}, ts2={ts2}, ts3={ts3}, ts4={ts4}, ts5={ts5}, ts6={ts6}");
105 }

```



```

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, 7
- ./Platform.Converters/CachingConverterDecorator.cs, 1
- ./Platform.Converters/CheckedConverter.cs, 1
- ./Platform.Converters/IConverter[TSource, TTarget].cs, 3
- ./Platform.Converters/IConverter[T].cs, 3
- ./Platform.Converters/To.cs, 4
- ./Platform.Converters/To[T].cs, 5
- ./Platform.Converters/UncheckedConverter.cs, 6