

LinksPlatform's Platform.Converters Class Library

1.1 ./Platform.Converters/CachingConverterDecorator.cs

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

1.2 ./Platform.Converters/CheckedConverter.cs

```
1 using System;
2 using System.Runtime.CompilerServices;
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 abstract class CheckedConverter<TSource, TTarget> : ConverterBase<TSource, TTarget>
10     {
11         public static CheckedConverter<TSource, TTarget> Default
12         {
13             [MethodImpl(MethodImplOptions.AggressiveInlining)]
14             get;
15         } = CompileCheckedConverter();
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         private static CheckedConverter<TSource, TTarget> CompileCheckedConverter()
19         {
20             var type = CreateTypeInheritedFrom<CheckedConverter<TSource, TTarget>>();
21             type.EmitFinalVirtualMethod<Converter<TSource, TTarget>>(nameof(IConverter<TSource,
22                 ↪ TTarget>.Convert), il =>
23             {
24                 il.LoadArgument(1);
25                 if (typeof(TSource) == typeof(object) && typeof(TTarget) != typeof(object))
26                 {
27                     ConvertAndUnbox(il);
28                 }
29                 else if (typeof(TSource) != typeof(object) && typeof(TTarget) != typeof(object))
30                 {
31                     il.CheckedConvert<TSource, TTarget>();
32                 }
33                 else if (typeof(TSource) != typeof(object) && typeof(TTarget) == typeof(object))
34                 {
35                     il.Box(typeof(TSource));
36                 }
37                 il.Return();
38             });
39             return (CheckedConverter<TSource,
40                 ↪ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
41         }
42     }
43 }
```

1.3 ./Platform.Converters/ConverterBase.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 ConverterBase<TSource, TTarget> : IConverter<TSource, TTarget>
12     {
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public abstract TTarget Convert(TSource source);
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         protected static void ConvertAndUnbox(ILGenerator il)
18         {
19             var typeContainer =
20                 ↪ typeof(NumericType<TTarget>).GetField(nameof(NumericType<TTarget>.Type),
21                 ↪ BindingFlags.Static | BindingFlags.Public);
22             il.Emit(OpCodes.Ldsfld, typeContainer);
23             il.Call(typeof(Convert).GetMethod(nameof(System.Convert.ChangeType), Types<object,
24                 ↪ Type>.Array));
25             il.UnboxValue(typeof(TTarget));
26         }
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         protected static string GetNewName() => Guid.NewGuid().ToString("N");
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         protected static TypeBuilder CreateTypeInheritedFrom<TBaseClass>()
33         {
34             var assemblyName = new AssemblyName(GetNewName());
35             var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,
36                 ↪ AssemblyBuilderAccess.Run);
37             var module = assembly.DefineDynamicModule(GetNewName());
38             var type = module.DefineType(GetNewName(), TypeAttributes.Public |
39                 ↪ TypeAttributes.Class | TypeAttributes.Sealed, typeof(TBaseClass));
40             return type;
41         }
42     }
43 }

```

1.4 ./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 }

```

1.5 ./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
8     ↪ значения.</para></typeparam>

```

```

8     public interface IConverter<T> : IConverter<T, T>
9     {
10    }
11 }

```

1.6 ./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
18             ↳ : (long)value);
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public static uint UInt32(ulong value) => unchecked(value > uint.MaxValue ?
22             ↳ uint.MaxValue : (uint)value);
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         public static int Int32(ulong value) => unchecked(value > int.MaxValue ? int.MaxValue :
26             ↳ (int)value);
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public static ushort UInt16(ulong value) => unchecked(value > ushort.MaxValue ?
30             ↳ ushort.MaxValue : (ushort)value);
31
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public static short Int16(ulong value) => unchecked(value > (ulong)short.MaxValue ?
34             ↳ short.MaxValue : (short)value);
35
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public static byte Byte(ulong value) => unchecked(value > byte.MaxValue ? byte.MaxValue
38             ↳ : (byte)value);
39
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public static sbyte SByte(ulong value) => unchecked(value > (ulong)sbyte.MaxValue ?
42             ↳ sbyte.MaxValue : (sbyte)value);
43
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public static bool Boolean(ulong value) => value > 0UL;
46
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public static char Char(ulong value) => unchecked(value > char.MaxValue ?
49             ↳ UnknownCharacter : (char)value);
50
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         public static DateTime DateTime(ulong value) => unchecked(value > long.MaxValue ?
53             ↳ System.DateTime.MaxValue : new DateTime((long)value));
54
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         public static TimeSpan TimeSpan(ulong value) => unchecked(value > long.MaxValue ?
57             ↳ System.TimeSpan.MaxValue : new TimeSpan((long)value));
58
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public static ulong UInt64(long value) => unchecked(value < (long)ulong.MinValue ?
61             ↳ ulong.MinValue : (ulong)value);
62
63         [MethodImpl(MethodImplOptions.AggressiveInlining)]
64         public static ulong UInt64(int value) => unchecked(value < (int)ulong.MinValue ?
65             ↳ ulong.MinValue : (ulong)value);
66
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         public static ulong UInt64(short value) => unchecked(value < (short)ulong.MinValue ?
69             ↳ ulong.MinValue : (ulong)value);
70
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         public static ulong UInt64(sbyte value) => unchecked(value < (sbyte)ulong.MinValue ?
73             ↳ ulong.MinValue : (ulong)value);
74     }
75 }

```

```

60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     public static ulong UInt64(bool value) => value ? 1UL : 0UL;
62
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public static ulong UInt64(char value) => value;
65
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
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
100 }

```

1.7 ./Platform.Converters/To[T].cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Exceptions;
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     [Obsolete]
11     public static class To<T>
12     {
13         public static readonly Func<T, object> Signed = CompileSignedDelegate();
14         public static readonly Func<T, object> Unsigned = CompileUnsignedDelegate();
15         public static readonly Func<object, T> UnsignedAs = CompileUnsignedAsDelegate();
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         static private Func<T, object> CompileSignedDelegate()
19         {
20             return DelegateHelpers.Compile<Func<T, object>>(emitter =>
21             {
22                 Ensure.Always.IsUnsignedInteger<T>();
23                 emitter.LoadArgument(0);
24                 var method = typeof(To).GetMethod("Signed", Types<T>.Array);
25                 emitter.Call(method);
26                 emitter.Box(method.ReturnType);
27                 emitter.Return();
28             });
29         }
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         static private Func<T, object> CompileUnsignedDelegate()
33         {
34             return DelegateHelpers.Compile<Func<T, object>>(emitter =>
35             {
36                 Ensure.Always.IsSignedInteger<T>();
37                 emitter.LoadArgument(0);

```

```

38         var method = typeof(To).GetMethod("Unsigned", Types<T>.Array);
39         emitter.Call(method);
40         emitter.Box(method.ReturnType);
41         emitter.Return();
42     });
43 }
44
45 [MethodImpl(MethodImplOptions.AggressiveInlining)]
46 static private Func<object, T> CompileUnsignedAsDelegate()
47 {
48     return DelegateHelpers.Compile<Func<object, T>>(emitter =>
49     {
50         Ensure.Always.IsUnsignedInteger<T>();
51         emitter.LoadArgument(0);
52         var signedVersion = NumericType<T>.SignedVersion;
53         emitter.UnboxValue(signedVersion);
54         var method = typeof(To).GetMethod("Unsigned", new[] { signedVersion });
55         emitter.Call(method);
56         emitter.Return();
57     });
58 }
59 }
60 }

```

1.8 ./Platform.Converters/UncheckedConverter.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
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 abstract class UncheckedConverter<TSource, TTarget> : ConverterBase<TSource, TTarget>
10     {
11         public static UncheckedConverter<TSource, TTarget> Default
12         {
13             [MethodImpl(MethodImplOptions.AggressiveInlining)]
14             get;
15             } = CompileUncheckedConverter();
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         private static UncheckedConverter<TSource, TTarget> CompileUncheckedConverter()
19         {
20             var type = CreateTypeInheritedFrom<UncheckedConverter<TSource, TTarget>>();
21             type.EmitFinalVirtualMethod<Converter<TSource,
22                 ↪ TTarget>>(nameof(IConverter<TSource, TTarget>.Convert), il =>
23             {
24                 il.LoadArgument(1);
25                 if (typeof(TSource) == typeof(object) && typeof(TTarget) != typeof(object))
26                 {
27                     ConvertAndUnbox(il);
28                 }
29                 else if (typeof(TSource) != typeof(object) && typeof(TTarget) != typeof(object))
30                 {
31                     il.UncheckedConvert<TSource, TTarget>();
32                 }
33                 else if (typeof(TSource) != typeof(object) && typeof(TTarget) == typeof(object))
34                 {
35                     il.Box(typeof(TSource));
36                 }
37                 il.Return();
38             });
39             return (UncheckedConverter<TSource,
40                 ↪ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
41         }
42     }
43 }

```

1.9 ./Platform.Converters.Tests/ConverterTests.cs

```

1  using Xunit;
2
3  namespace Platform.Converters.Tests
4  {
5      public class ConverterTests
6      {
7          [Fact]
8          public void SameTypeTest()
9          {

```

```
10     var result = UncheckedConverter<ulong, ulong>.Default.Convert(2UL);
11     Assert.Equal(2UL, result);
12     result = CheckedConverter<ulong, ulong>.Default.Convert(2UL);
13     Assert.Equal(2UL, result);
14 }
15
16 [Fact]
17 public void Int32ToUInt64Test()
18 {
19     var result = UncheckedConverter<int, ulong>.Default.Convert(2);
20     Assert.Equal(2UL, result);
21     result = CheckedConverter<int, ulong>.Default.Convert(2);
22     Assert.Equal(2UL, result);
23 }
24 }
25 }
```

Index

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