

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             EmitConvertMethod(type, il => il.CheckedConvert<TSource, TTarget>());
22             return (CheckedConverter<TSource,
23                 ↪ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
24         }
25     }
26 }
```

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         }
23     }
24 }
```

```

20         il.Emit(OpCodes.Ldsfld, typeContainer);
21         il.Call(typeof(Convert).GetMethod(nameof(System.Convert.ChangeType), Types<object,
        ↳ Type>.Array));
22         il.UnboxValue(typeof(TTarget));
23     }
24
25     [MethodImpl(MethodImplOptions.AggressiveInlining)]
26     protected static string GetNewName() => Guid.NewGuid().ToString("N");
27
28     [MethodImpl(MethodImplOptions.AggressiveInlining)]
29     protected static TypeBuilder CreateTypeInheritedFrom<TBaseClass>()
30     {
31         var assemblyName = new AssemblyName(GetNewName());
32         var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,
        ↳ AssemblyBuilderAccess.Run);
33         var module = assembly.DefineDynamicModule(GetNewName());
34         var type = module.DefineType(GetNewName(), TypeAttributes.Public |
        ↳ TypeAttributes.Class | TypeAttributes.Sealed, typeof(TBaseClass));
35         return type;
36     }
37
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     protected static void EmitConvertMethod(TypeBuilder typeBuilder, Action<ILGenerator>
        ↳ emitConversion)
40     {
41         typeBuilder.EmitFinalVirtualMethod<Converter<TSource,
        ↳ TTarget>>(nameof(IConverter<TSource, TTarget>.Convert), il =>
42         {
43             il.LoadArgument(1);
44             if (typeof(TSource) == typeof(object) && typeof(TTarget) != typeof(object))
45             {
46                 ConvertAndUnbox(il);
47             }
48             else if (typeof(TSource) != typeof(object) && typeof(TTarget) == typeof(object))
49             {
50                 il.Box(typeof(TSource));
51             }
52             else
53             {
54                 emitConversion(il);
55             }
56             il.Return();
57         });
58     }
59 }
60 }

```

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
        ↳ 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 }

```

1.5 ./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
    ↪ 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
    ↪ : (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 }

```

1.7 ./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 = CreateTypeInfoInheritedFrom<UncheckedConverter<TSource, TTarget>>();
21                 EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>());
22                 return (UncheckedConverter<TSource,
23                     ↪ TTarget>) Activator.CreateInstance(type.CreateTypeInfo());
24             }
25 }

```

1.9 ./Platform.Converters/UncheckedSignExtendingConverter.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 UncheckedSignExtendingConverter<TSource, TTarget> :
10     ↪ ConverterBase<TSource, TTarget>
11     {
12         public static UncheckedSignExtendingConverter<TSource, TTarget> Default
13         {
14             [MethodImpl(MethodImplOptions.AggressiveInlining)]
15             get;
16             } = CompileUncheckedConverter();
17
18             [MethodImpl(MethodImplOptions.AggressiveInlining)]
19             private static UncheckedSignExtendingConverter<TSource, TTarget>
20             ↪ CompileUncheckedConverter()

```

```

19     {
20         var type = CreateTypeInfoInheritedFrom<UncheckedSignExtendingConverter<TSource,
21             ↳ TTarget>>>();
22         EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>(extendSign:
23             ↳ true));
24         return (UncheckedSignExtendingConverter<TSource,
25             ↳ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
26     }
27 }
28 }

```

1.10 ./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         [Fact]
26         public void SignExtensionTest()
27         {
28             var result = UncheckedSignExtendingConverter<byte, long>.Default.Convert(128);
29             Assert.Equal(-128L, result);
30             result = UncheckedConverter<byte, long>.Default.Convert(128);
31             Assert.Equal(128L, result);
32         }
33     }
34 }

```

Index

- ./Platform.Converters.Tests/ConverterTests.cs, 6
- ./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
- ./Platform.Converters/UncheckedSignExtendingConverter.cs, 5