./To.cs

```
using System;
using System.Runtime.CompilerServices;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Converters
{
    public static class To
    {
        public static readonly char UnknownCharacter = '';

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong UInt64(ulong value) => value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static long Int64(ulong value) => value > long.MaxValue ? long.MaxValue :
        ↪  (long)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static uint UInt32(ulong value) => value > uint.MaxValue ? uint.MaxValue :
        ↪  (uint)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static int Int32(ulong value) => value > int.MaxValue ? int.MaxValue : (int)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ushort UInt16(ulong value) => value > ushort.MaxValue ? ushort.MaxValue :
        ↪  (ushort)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static short Int16(ulong value) => value > (ulong)short.MaxValue ? short.MaxValue
        ↪  : (short)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static byte Byte(ulong value) => value > byte.MaxValue ? byte.MaxValue :
        ↪  (byte)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static sbyte SByte(ulong value) => value > (ulong)sbyte.MaxValue ? sbyte.MaxValue
        ↪  : (sbyte)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool Boolean(ulong value) => value > 0;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static char Char(ulong value) => value > char.MaxValue ? UnknownCharacter :
        ↪  (char)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static DateTime DateTime(ulong value) => value > long.MaxValue ?
        ↪  System.DateTime.MaxValue : new DateTime((long)value);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TimeSpan TimeSpan(ulong value) => value > long.MaxValue ?
        ↪  System.TimeSpan.MaxValue : new TimeSpan((long)value);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong UInt64(long value) => value < (long)ulong.MinValue ? ulong.MinValue
        ↪  : (ulong)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong UInt64(int value) => value < (int)ulong.MinValue ? ulong.MinValue :
        ↪  (ulong)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong UInt64(short value) => value < (short)ulong.MinValue ?
        ↪  ulong.MinValue : (ulong)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong UInt64(sbyte value) => value < (sbyte)ulong.MinValue ?
        ↪  ulong.MinValue : (ulong)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong UInt64(bool value) => value ? 1UL : 0UL;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
        public static ulong UInt64(char value) => value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static long Signed(ulong value) => (long)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static int Signed(uint value) => (int)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static short Signed(ushort value) => (short)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static sbyte Signed(byte value) => (sbyte)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static object Signed<T>(T value) => To<T>.Signed(value);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong Unsigned(long value) => (ulong)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static uint Unsigned(int value) => (uint)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ushort Unsigned(short value) => (ushort)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static byte Unsigned(sbyte value) => (byte)value;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static object Unsigned<T>(T value) => To<T>.Unsigned(value);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T UnsignedAs<T>(object value) => To<T>.UnsignedAs(value);
    }
}
```

./To[T].cs

```
using System;
using System.Reflection;
using Platform.Exceptions;
using Platform.Reflection;
using Platform.Reflection.Sigil;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Converters
{
    public sealed class To<T>
    {
        public static readonly Func<T, object> Signed;
        public static readonly Func<T, object> Unsigned;
        public static readonly Func<object, T> UnsignedAs;

        static To()
        {
            Signed = DelegateHelpers.Compile<Func<T, object>>(emiter =>
            {
                Ensure.Always.IsUnsignedInteger<T>();
                emiter.LoadArgument(0);
                var method = typeof(To).GetTypeInfo().GetMethod("Signed", Types<T>.Array);
                emiter.Call(method);
                emiter.Box(method.ReturnType);
                emiter.Return();
            });
            Unsigned = DelegateHelpers.Compile<Func<T, object>>(emiter =>
            {
                Ensure.Always.IsSignedInteger<T>();
                emiter.LoadArgument(0);
                var method = typeof(To).GetTypeInfo().GetMethod("Unsigned", Types<T>.Array);
                emiter.Call(method);
                emiter.Box(method.ReturnType);
                emiter.Return();
            });
            UnsignedAs = DelegateHelpers.Compile<Func<object, T>>(emiter =>
            {
                Ensure.Always.IsUnsignedInteger<T>();
                emiter.LoadArgument(0);
                var signedVersion = CachedTypeInfo<T>.SignedVersion;
                emiter.UnboxAny(signedVersion);
```

```
                var method = typeof(To).GetTypeInfo().GetMethod("Unsigned", new[] {
                ↪  signedVersion });
                emiter.Call(method);
                emiter.Return();
            });
        }

        private To()
        {
        }
    }
}
```

Index