```
LinksPlatform's Platform Converters Class Library
     ./csharp/Platform.Converters/CachingConverterDecorator.cs
   using System.Collections.Generic;
using System.Runtime.CompilerServices;
2
   using Platform.Collections;
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform.Converters
        /// <summary>
9
        /// <para>
10
        /// Represents the caching converter decorator.
11
        /// </para>
12
        /// <para></para>
13
        /// <\bar{\summary>
        /// <seealso cref="IConverter{TSource, TTarget}"/>
public class CachingConverterDecorator<TSource, TTarget> : IConverter<TSource, TTarget>
15
17
            private readonly IConverter<TSource, TTarget> _baseConverter;
private readonly IDictionary<TSource, TTarget> _cache;
18
19
20
             /// <summary>
21
            /// <para>
22
             /// Initializes a new <see cref="CachingConverterDecorator"/> instance.
24
             /// </para>
            /// <para></para>
25
            /// </summary>
26
            /// <param name="baseConverter">
27
            /// <para>A base converter.</para>
2.8
            /// <para></para>
29
             /// </param>
             /// <param name="cache">
31
             /// <para>A cache.</para>
32
             /// <para></para>
33
             /// </param>
34
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
35
            public CachingConverterDecorator(IConverter<TSource, TTarget> baseConverter,
36
                IDictionary<TSource, TTarget> cache) => (_baseConverter, _cache) = (baseConverter,
                cache);
37
             /// <summary>
38
39
            /// Initializes a new <see cref="CachingConverterDecorator"/> instance.
40
            /// </para>
41
            /// <para></para>
             /// </summary>
             /// <param name="baseConverter">
44
             /// <para>A base converter.</para>
45
             /// <para></para>
46
             /// </param>
47
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
48
            public CachingConverterDecorator(IConverter<TSource, TTarget> baseConverter) :
49
                this(baseConverter, new Dictionary<TSource, TTarget>()) { }
50
             /// <summary>
5.1
             /// <para>
            /// Converts the source.
53
            /// </para>
54
            /// <para></para>
55
             /// </summary>
            /// <param name="source">
57
             /// <para>The source.</para>
58
             /// <para></para>
            /// </param>
60
            /// <returns>
61
            /// <para>The target</para>
62
             /// <para></para>
             /// </returns>
64
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
65
            public TTarget Convert(TSource source) => _cache.GetOrAdd(source,
             }
67
68
```

## 1.2 ./csharp/Platform.Converters/CheckedConverter.cs using System; using System.Runtime.CompilerServices;

```
using Platform.Reflection;
3
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform.Converters
7
        /// <summary>
9
       /// <para>
10
       /// Represents the checked converter.
       /// </para>
12
       /// <para></para>
13
       /// </summary>
14
       /// <seealso cref="ConverterBase{TSource, TTarget}"/>
15
       public abstract class CheckedConverter<TSource, TTarget> : ConverterBase<TSource, TTarget>
16
17
            /// <summary>
18
            /// <para>
19
            /// Gets the default value.
20
            /// </para>
21
            /// <para></para>
22
            /// </summary>
23
           public static CheckedConverter<TSource, TTarget> Default
24
25
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
27
            } = CompileCheckedConverter();
29
            /// <summary>
30
            /// <para>
31
            /// Compiles the checked converter.
32
            /// </para>
33
            /// <para></para>
            /// </summary>
35
            /// <returns>
36
            /// <para>A checked converter of t source and t target</para>
37
            /// <para></para>
            /// </returns>
39
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
40
           private static CheckedConverter<TSource, TTarget> CompileCheckedConverter()
42
                var type = CreateTypeInheritedFrom<CheckedConverter<TSource, TTarget>>();
43
                EmitConvertMethod(type, il => il.CheckedConvert<TSource, TTarget>());
44
                return (CheckedConverter<TSource,
45
                → TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
           }
46
       }
47
   }
48
    ./csharp/Platform.Converters/ConverterBase.cs
   using System;
   using System. Reflection;
   using System.Reflection.Emit;
using System.Runtime.CompilerServices;
3
4
   using Platform.Reflection;
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
   namespace Platform.Converters
9
10
11
        /// <summary>
       /// <para>Represents a base implementation for IConverter interface with the basic logic
12
        __ necessary for value converter from the <typeparamref name="TSource"/> type to the
           <typeparamref name="TTarget"/> type.</para>
       /// <para>Представляет базовую реализацию для интерфейса IConverter с основной логикой
13
           необходимой для конвертера значений из типа <typeparamref name="TSource"/> в тип
           <typeparamref name="TTarget"/>.</para>
       /// </summary>
14
       /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
15
          конверсии.</para></typeparam>
        /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
           конверсии.</para></typeparam>
       public abstract class ConverterBase<TSource, TTarget> : IConverter<TSource, TTarget>
17
18
            /// <summary>
19
            /// <para>Converts the value of the <typeparamref name="TSource"/> type to the value of
20
               the <typeparamref name="TTarget"/> type.</para>
            /// <para>Kонвертирует значение типа <typeparamref name="TSource"/> в значение типа
```

```
/// </summary>
            /// <param name="source"><para>The <typeparamref name=="TSource"/> type
                value.</para><para>Значение типа <typeparamref name="TSource"/>.</para></param>
            /// <returns><para>The converted value of the <typeparamref name="TTarget"/>
               type.</para><para>Значение конвертированное в тип <typeparamref
               name="TTarget"/>.</para></returns>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
25
           public abstract TTarget Convert(TSource source);
26
27
            /// <summary>
2.8
            /// <para>Generates a sequence of instructions using <see cref="ILGenerator"/> that
               converts a value of type <see cref="System.Object"/> to a value of type
               <typeparamref name="TTarget"/>.</para>
            /// <para>Генерирует последовательность инструкций при помощи <see cref="ILGenerator"/>
30
               выполняющую преобразование значения типа <see cref="System.Object"/> к значению типа
               <typeparamref name="TTarget"/>.</para>
            /// </summary>
            /// <param name="il"><para>An <see cref="ILGenerator"/> instance.</para><para>Экземпляр
32
               <see cref="ILGenerator"/>.</para></param>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
           protected static void ConvertFromObject(ILGenerator il)
34
35
                var returnDefault = il.DefineLabel();
                il.Emit(OpCodes.Brfalse_S, returnDefault);
                il.LoadArgument(1);
38
                il.Emit(OpCodes.Castclass, typeof(IConvertible));
39
                il.Emit(OpCodes.Ldnull);
                il.Emit(OpCodes.Callvirt, GetMethodForConversionToTargetType());
41
                il.Return()
42
                il.MarkLabel(returnDefault);
43
                LoadDefault(il, typeof(TTarget));
44
            /// <summary>
47
            /// <para>Gets a new unique name of an assembly.</para>
48
            /// <para>Возвращает новое уникальное имя сборки.</para>
            /// </summary>
50
            /// <returns><para>A new unique name of an assembly.</para><para>Новое уникальное имя
               сборки.</para></returns>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
52
           protected static string GetNewName() => Guid.NewGuid().ToString("N");
54
            /// <summary>
            /// <para>Converts the value of the source type (TSource) to the value of the target
56
               type.</para>
            /// <para>Kонвертирует значение исходного типа (TSource) в значение целевого типа.</para>
            /// <\(\bar{\summary}\)
            /// <param name="source"><para>The source type value (TSource).</para><para>Значение
               исходного типа (TSource).</para></param>
            /// <returns><para>The value is converted to the target type
60
               (TTarget).</para><para>Значение ковертированное в целевой тип
                (TTarget).</para></returns>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
           protected static TypeBuilder CreateTypeInheritedFrom<TBaseClass>()
                var assemblyName = new AssemblyName(GetNewName());
                var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,

→ AssemblyBuilderAccess.Run);
                var module = assembly.DefineDynamicModule(GetNewName());
                var type = module.DefineType(GetNewName(), TypeAttributes.Public |
67
                   TypeAttributes.Class | TypeAttributes.Sealed, typeof(TBaseClass));
                return type;
           }
7.0
            /// <summary>
            /// <para>Converts the value of the source type (TSource) to the value of the target
               type.</para>
            /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
7.3
            /// </summary>
            /// <param name="source"><para>The source type value (TSource).</para><para>Значение
               исходного типа (TSource).</para></param>
            /// <returns><para>The value is converted to the target type
                (TTarget).</para><para>Значение ковертированное в целевой тип
                (TTarget).</para></returns>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
           protected static void EmitConvertMethod(TypeBuilder typeBuilder, Action<ILGenerator>
               emitConversion)
```

```
typeBuilder.EmitFinalVirtualMethod<Converter<TSource,
        TTarget>>(nameof(IConverter<TSource, TTarget>.Convert), il =>
        il.LoadArgument(1);
        if (typeof(TSource) == typeof(object) && typeof(TTarget) != typeof(object))
            ConvertFromObject(il);
        else if (typeof(TSource) != typeof(object) && typeof(TTarget) == typeof(object))
            il.Box(typeof(TSource));
        }
        else
        {
            emitConversion(il);
        il.Return();
    });
}
/// <summary>
/// <para>Converts the value of the source type (TSource) to the value of the target
    type.</para>
/// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
/// </summary>
/// <param name="source"><para>The source type value (TSource).</para><para>Значение
   исходного типа (TSource).</para></param>
/// <returns><para>The value is converted to the target type
   (TTarget).</para><para>Значение ковертированное в целевой тип
    (TTarget).</para></returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected static MethodInfo GetMethodForConversionToTargetType()
    var targetType = typeof(TTarget);
    var convertibleType = typeof(IConvertible);
    var typeParameters = Types<IFormatProvider>.Array;
    if (targetType == typeof(bool))
        return convertibleType.GetMethod(nameof(IConvertible.ToBoolean), typeParameters);
    }
    else if (targetType == typeof(byte))
        return convertibleType.GetMethod(nameof(IConvertible.ToByte), typeParameters);
    else if (targetType == typeof(char))
        return convertibleType.GetMethod(nameof(IConvertible.ToChar), typeParameters);
    else if (targetType == typeof(DateTime))
        return convertibleType.GetMethod(nameof(IConvertible.ToDateTime),

→ typeParameters);

    }
    else if (targetType == typeof(decimal))
        return convertibleType.GetMethod(nameof(IConvertible.ToDecimal), typeParameters);
    }
    else if (targetType == typeof(double))
        return convertibleType.GetMethod(nameof(IConvertible.ToDouble), typeParameters);
    }
    else if (targetType == typeof(short))
        return convertibleType.GetMethod(nameof(IConvertible.ToInt16), typeParameters);
    else if (targetType == typeof(int))
        return convertibleType.GetMethod(nameof(IConvertible.ToInt32), typeParameters);
    else if (targetType == typeof(long))
        return convertibleType.GetMethod(nameof(IConvertible.ToInt64), typeParameters);
    else if (targetType == typeof(sbyte))
        return convertibleType.GetMethod(nameof(IConvertible.ToSByte), typeParameters);
```

80

82

84

85

87 88

89

91 92

93 94

95

96

97 98

99

102

103

104

105

106 107

108

109

110

112

113

114

115 116

117 118

119 120

122

123 124

126

127 128

129

130

131

133

134

135 136

137 138

139 140

141

 $\frac{143}{144}$ 

145

147 148

149

```
150
                 else if (targetType == typeof(float))
152
                     return convertibleType.GetMethod(nameof(IConvertible.ToSingle), typeParameters);
153
                 else if (targetType == typeof(string))
155
156
                     return convertibleType.GetMethod(nameof(IConvertible.ToString), typeParameters);
157
                 else if (targetType == typeof(ushort))
159
160
                     return convertibleType.GetMethod(nameof(IConvertible.ToUInt16), typeParameters);
161
                 }
162
                 else if (targetType == typeof(uint))
163
164
                     return convertibleType.GetMethod(nameof(IConvertible.ToUInt32), typeParameters);
                 }
166
                 else if (targetType == typeof(ulong))
167
168
                     return convertibleType.GetMethod(nameof(IConvertible.ToUInt64), typeParameters);
169
                 }
170
                 else
171
                 {
172
                     throw new NotSupportedException();
173
                 }
             }
175
176
             /// <summary>
177
             /// <para>Converts the value of the source type (TSource) to the value of the target
178
                 type.</para>
             /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
179
             /// </summary>
180
             /// <param name="source"><para>The source type value (TSource).</para><para>Значение
181
             → исходного типа (TSource).</para></param>
             /// <returns><para>The value is converted to the target type
182
                 (TTarget).</para>Значение ковертированное в целевой тип
                 (TTarget).</para></returns>
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
183
            protected static void LoadDefault(ILGenerator il, Type targetType)
185
                 if (targetType == typeof(string))
186
187
                     il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(string.Empty),
188

→ BindingFlags.Static | BindingFlags.Public));
189
                 else if (targetType == typeof(DateTime))
190
                     il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(DateTime.MinValue),
192
                      → BindingFlags.Static | BindingFlags.Public));
193
                 else if (targetType == typeof(decimal))
194
                     il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(decimal.Zero),
196
                         BindingFlags.Static | BindingFlags.Public));
197
                 else if (targetType == typeof(float))
198
199
                     il.LoadConstant(0.0F);
200
                 }
201
                 else if (targetType == typeof(double))
203
                     il.LoadConstant(0.0D);
204
                 }
205
                 else if (targetType == typeof(long) || targetType == typeof(ulong))
206
207
                     il.LoadConstant(OL);
208
                 }
209
                 else
210
211
                     il.LoadConstant(0);
212
                 }
213
            }
214
        }
215
    }
216
```

```
./csharp/Platform.Converters/IConverter[TSource, TTarget].cs
   namespace Platform.Converters
2
        /// <summary>
3
       /// <para>Defines a value converter from the <typeparamref name="TSource"/> type to the
4
           <typeparamref name="TTarget"/> type.</para>
        /// <para>Определяет конвертер значений из типа <typeparamref name="TSource"/> в тип
           <typeparamref name="TTarget"/>.</para>
        /// </summary>
       /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
           конверсии.</para></typeparam>
        /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
           конверсии.</para></typeparam>
       public interface IConverter<in TSource, out TTarget>
10
            /// <summary>
11
            /// <para>Converts the value of the <typeparamref name="TSource"/> type to the value of
12
               the <typeparamref name="TTarget"/> type.</para>
            /// <para>Конвертирует значение типа <typeparamref name="TSource"/> в значение типа
13
               <typeparamref name="TTarget"/>.</para>
            /// </summary>
14
            /// <param name="source"><para>The <typeparamref name=="TSource"/> type
15
                value.</para><para>Значение типа <typeparamref name="TSource"/>.</para></param>
            /// <returns><para>The converted value of the <typeparamref name="TTarget"/>
               type.</para><para>Значение конвертированное в тип <typeparamref
            → name="TTarget"/>.</para></returns>
            TTarget Convert(TSource source);
       }
18
   }
     ./csharp/Platform.Converters/IConverter[T].cs
   namespace Platform.Converters
2
        /// <summary>
3
       /// <para>Defines a converter between two values of the same <typeparamref name="T"/>
4
           type.</para>
       /// <para>Определяет конвертер между двумя значениями одного типа <typeparamref
           name="T"/>.</para>
       /// </summary>
       /// <typeparam name="T"><para>The type of value to convert.</para><para>Тип преобразуемого
           значения.</para></typeparam>
       public interface IConverter<T> : IConverter<T, T>
9
10
11
     ./csharp/Platform.Converters/UncheckedConverter.cs
   using System;
   using System.Runtime.CompilerServices;
   using Platform.Reflection;
3
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
   namespace Platform.Converters
7
        /// <summary>
9
        /// <para>
10
        /// Represents the unchecked converter.
11
       /// </para>
12
       /// <para></para>
13
       /// </summary>
14
        /// <seealso cref="ConverterBase{TSource, TTarget}"/>
       public abstract class UncheckedConverter<TSource, TTarget> : ConverterBase<TSource, TTarget>
16
17
            /// <summary>
            /// <para>
19
            /// Gets the default value.
            /// </para>
21
            /// <para></para>
22
            /// </summary
23
           public static UncheckedConverter<TSource, TTarget> Default
24
25
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
27
            } = CompileUncheckedConverter();
28
29
            /// <summary>
30
```

```
/// <para>
31
            /// Compiles the unchecked converter.
            /// </para>
33
            /// <para></para>
34
            /// </summary>
            /// <returns>
            /// <para>An unchecked converter of t source and t target</para>
37
            /// <para></para>
38
            /// </returns>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
40
            private static UncheckedConverter<TSource, TTarget> CompileUncheckedConverter()
41
42
                var type = CreateTypeInheritedFrom<UncheckedConverter<TSource, TTarget>>();
                EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>());
44
                return (UncheckedConverter<TSource,
45
                → TTarget>) Activator. CreateInstance(type.CreateTypeInfo());
            }
46
       }
47
   }
48
     ./csharp/Platform.Converters/UncheckedSignExtendingConverter.cs
1.7
   using System;
   using System.Runtime.CompilerServices;
   using Platform.Reflection;
3
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
   namespace Platform.Converters
        /// <summary>
9
        /// <para>
10
        /// Represents the unchecked sign extending converter.
11
        /// </para>
12
        /// <para></para>
13
        /// </summary>
14
        /// <seealso cref="ConverterBase{TSource, TTarget}"/>
15
       public abstract class UncheckedSignExtendingConverter<TSource, TTarget> :
16
           ConverterBase<TSource, TTarget>
            /// <summary>
18
            /// <para>
19
            /// Gets the default value.
20
            /// </para>
21
            /// <para></para>
22
            /// </summary>
23
            public static UncheckedSignExtendingConverter<TSource, TTarget> Default
24
25
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
27
            } = CompileUncheckedConverter();
29
30
            /// <summary>
            /// <para>
            /// Compiles the unchecked converter.
32
            /// </para>
33
            /// <para></para>
            /// </summary>
35
            /// <returns>
36
            /// <para>An unchecked sign extending converter of t source and t target</para>
37
            /// <para></para>
38
            /// </returns>
39
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
40
            private static UncheckedSignExtendingConverter<TSource, TTarget>
41
               CompileUncheckedConverter()
42
                var type = CreateTypeInheritedFrom<UncheckedSignExtendingConverter<TSource,</pre>
                    TTarget>>();
                EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>(extendSign:
44

    true));
                return (UncheckedSignExtendingConverter<TSource,</pre>
45
                 → TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
            }
       }
    ./csharp/Platform.Converters.Tests/ConverterTests.cs
```

using System;
using Xunit;

```
namespace Platform.Converters.Tests
4
5
        /// <summary>
        /// <para>
7
        /// Represents the converter tests.
8
        /// </para>
9
        /// <para></para>
10
        /// </summary>
11
        public static class ConverterTests
12
13
            /// <summary>
14
15
            /// <para>
            /// Tests that same type test.
16
            /// </para>
17
            /// <para></para>
18
            /// </summary>
            [Fact]
20
            public static void SameTypeTest()
21
22
                var result = UncheckedConverter<ulong, ulong>.Default.Convert(2UL);
23
                Assert.Equal(2UL, result);
24
                result = CheckedConverter<ulong, ulong>.Default.Convert(2UL);
25
                Assert.Equal(2UL, result);
            }
27
28
            /// <summary>
29
            /// <para>
30
            /// Tests that int 32 to u int 64 test.
            /// </para>
            /// <para></para>
33
            /// </summary>
34
35
            [Fact]
            public static void Int32ToUInt64Test()
36
37
                var result = UncheckedConverter<int, ulong>.Default.Convert(2);
38
                Assert.Equal(2UL, result);
                result = CheckedConverter<int, ulong>.Default.Convert(2);
40
                Assert.Equal(2UL, result);
41
            }
42
43
            /// <summary>
44
            /// <para>
            /// Tests that sign extension test.
46
            /// </para>
47
            /// <para></para>
            /// </summary>
49
            [Fact]
50
            public static void SignExtensionTest()
                var result = UncheckedSignExtendingConverter<byte, long>.Default.Convert(128);
53
                Assert.Equal(-128L, result);
54
55
                result = UncheckedConverter<byte, long>.Default.Convert(128);
                Assert.Equal(128L, result);
56
            }
57
            /// <summary>
59
            /// <para>
60
            /// Tests that object test.
61
            /// </para>
62
            /// <para></para>
63
            /// </summary>
            [Fact]
65
            public static void ObjectTest()
66
67
                TestObjectConversion("1");
68
                TestObjectConversion(DateTime.UtcNow);
69
                TestObjectConversion(1.0F);
70
71
                TestObjectConversion(1.0D);
                TestObjectConversion(1.0M);
72
                TestObjectConversion(1UL);
73
                TestObjectConversion(1L);
74
75
                TestObjectConversion(1U);
                TestObjectConversion(1);
76
                TestObjectConversion((char)1);
77
                TestObjectConversion((ushort)1);
78
                TestObjectConversion((short)1);
79
                TestObjectConversion((byte)1);
80
```

```
TestObjectConversion((sbyte)1);
TestObjectConversion(true);
}

private static void TestObjectConversion<T>(T value) => Assert.Equal(value,
UncheckedConverter<object, T>.Default.Convert(value));

UncheckedConverter<object, T>.Default.Convert(value));
```

## Index

```
./csharp/Platform.Converters.Tests/ConverterTests.cs, 7
./csharp/Platform.Converters/CachingConverterDecorator.cs, 1
./csharp/Platform.Converters/CheckedConverter.cs, 1
./csharp/Platform.Converters/ConverterBase.cs, 2
./csharp/Platform.Converters/IConverter[TSource, TTarget].cs, 5
./csharp/Platform.Converters/IConverter[T].cs, 6
./csharp/Platform.Converters/UncheckedConverter.cs, 6
./csharp/Platform.Converters/UncheckedSignExtendingConverter.cs, 7
```