```
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
        /// </summary>
        /// <seealso cref="IConverter{TSource, TTarget}"/>
public class CachingConverterDecorator<TSource, TTarget> : IConverter<TSource, TTarget>
15
16
17
            /// <summary>
18
            /// <para>
19
            /// The base converter.
            /// </para>
21
            /// <para></para>
22
            /// </summary>
23
            private readonly IConverter<TSource, TTarget> _baseConverter;
^{24}
            /// <summary>
            /// <para>
            /// The cache.
27
            /// </para>
28
            /// <para></para>
29
            /// </summary>
30
            private readonly IDictionary<TSource, TTarget> _cache;
32
            /// <summary>
33
            /// <para>
34
            /// Initializes a new <see cref="CachingConverterDecorator"/> instance.
35
            /// </para>
36
            /// <para></para>
            /// </summary>
38
            /// <param name="baseConverter">
39
            /// <para>A base converter.</para>
40
            /// <para></para>
41
            /// </param>
42
            /// <param name="cache">
43
            /// <para>A cache.</para>
            /// <para></para>
45
            /// </param>
46
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
47
            public CachingConverterDecorator(IConverter<TSource, TTarget> baseConverter,
                IDictionary<TSource, TTarget> cache) => (_baseConverter, _cache) = (baseConverter,
                cache);
49
            /// <summary>
            /// <para>
5.1
            /// Initializes a new <see cref="CachingConverterDecorator"/> instance.
52
            /// </para>
53
            /// <para></para>
54
            /// </summary>
55
            /// <param name="baseConverter">
56
            /// <para>A base converter.</para>
            /// <para></para>
58
            /// </param>
59
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            public CachingConverterDecorator(IConverter<TSource, TTarget> baseConverter) :
61
                this(baseConverter, new Dictionary<TSource, TTarget>()) { }
62
            /// <summary>
            /// <para>
64
            /// Converts the source.
65
            /// </para>
66
            /// <para></para>
67
            /// </summary>
68
            /// <param name="source">
            /// <para>The source.</para>
70
            /// <para></para>
7.1
            /// </param>
72
            /// <returns>
```

```
/// <para>The target</para>
74
           /// <para></para>
75
           /// </returns>
76
           [MethodImpl(MethodImplOptions.AggressiveInlining)]
77
           public TTarget Convert(TSource source) => _cache.GetOrAdd(source,
               _baseConverter.Convert);
       }
   }
80
    ./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
8
       /// <summary>
       /// <para>
10
       /// Represents the checked converter.
11
       /// </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>
           /// <para></para>
           /// </summary>
23
           public static CheckedConverter<TSource, TTarget> Default
24
25
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
26
27
           } = CompileCheckedConverter();
28
29
           /// <summary>
30
           /// <para>
31
           /// Compiles the checked converter.
           /// </para>
33
           /// <para></para>
/// </summary>
34
35
           /// <returns>
36
           /// <para>A checked converter of t source and t target</para>
37
           /// <para></para>
38
           /// </returns>
39
           [MethodImpl(MethodImplOptions.AggressiveInlining)]
40
           private static CheckedConverter<TSource, TTarget> CompileCheckedConverter()
41
42
               var type = CreateTypeInheritedFrom<CheckedConverter<TSource, TTarget>>();
               EmitConvertMethod(type, il => il.CheckedConvert<TSource, TTarget>());
44
               return (CheckedConverter<TSource,
45
                TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
           }
46
       }
47
48
1.3
   ./csharp/Platform.Converters/ConverterBase.cs
   using System;
         System Reflection;
   using
   using System.Reflection.Emit;
   using System.Runtime.CompilerServices;
   using Platform.Reflection;
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform.Converters
9
10
       /// <summary>
11
       /// <para>Represents a base implementation for IConverter interface with the basic logic
12
        <typeparamref name="TTarget"/> type.</para>
       /// <para>Представляет базовую реализацию для интерфейса IConverter с основной логикой
        🛶 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
            /// <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
21
               <typeparamref name="TTarget"/>.</para>
            /// </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)]
           public abstract TTarget Convert(TSource source);
26
27
            /// <summary>
           /// <para>Generates a sequence of instructions using <see cref="ILGenerator"/> that
29
               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)]
33
           protected static void ConvertFromObject(ILGenerator il)
34
                var returnDefault = il.DefineLabel();
36
                il.Emit(OpCodes.Brfalse_S, returnDefault);
37
                il.LoadArgument(1);
38
                il.Emit(OpCodes.Castclass, typeof(IConvertible));
39
                il.Emit(OpCodes.Ldnull);
40
                il.Emit(OpCodes.Callvirt, GetMethodForConversionToTargetType());
41
42
                il.Return()
                il.MarkLabel(returnDefault);
43
                LoadDefault(il, typeof(TTarget));
44
           }
46
            /// <summary>
47
            /// <para>Gets a new unique name of an assembly.</para>
           /// <para>Возвращает новое уникальное имя сборки.</para>
49
           /// </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");
53
            /// <summary>
55
            /// <para>Converts the value of the source type (TSource) to the value of the target
56
               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
5.9
               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>()
63
                var assemblyName = new AssemblyName(GetNewName());
               var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,

→ AssemblyBuilderAccess.Run);
                var module = assembly.DefineDynamicModule(GetNewName());
66
                var type = module.DefineType(GetNewName(), TypeAttributes.Public |
67
                   TypeAttributes.Class | TypeAttributes.Sealed, typeof(TBaseClass));
                return type;
           }
69
```

```
/// <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>
74
            /// <param name="source"><para>The source type value (TSource).</para><para>Значение
7.5
                исходного типа (TSource).</para></param>
            /// <returns><para>The value is converted to the target type
                (TTarget).</para><pаra>Значение ковертированное в целевой тип
                (TTarget).</para></returns>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected static void EmitConvertMethod(TypeBuilder typeBuilder, Action<ILGenerator>
78
                emitConversion)
                typeBuilder.EmitFinalVirtualMethod<Converter<TSource,
80
                     TTarget>>(nameof(IConverter<TSource, TTarget>.Convert), il =>
                     il.LoadArgument(1);
                     if (typeof(TSource) == typeof(object) && typeof(TTarget) != typeof(object))
83
                     {
84
                         ConvertFromObject(il);
85
                     }
                     else if (typeof(TSource) != typeof(object) && typeof(TTarget) == typeof(object))
87
88
                         il.Box(typeof(TSource));
90
                     else
91
92
                         emitConversion(il);
93
94
                     il.Return();
                });
96
            }
97
98
            /// <summary>
99
            /// <para>Converts the value of the source type (TSource) to the value of the target
100
                type.</para>
            /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
            /// </summary>
102
            /// <param name="source"><para>The source type value (TSource).</para><para>Значение
103
                исходного типа (TSource).</para></param>
            /// <returns><para>The value is converted to the target type
104
                (TTarget).</para><para>Значение ковертированное в целевой тип
                (TTarget).</para></returns>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected static MethodInfo GetMethodForConversionToTargetType()
106
107
                var targetType = typeof(TTarget);
108
                var convertibleType = typeof(IConvertible);
109
                var typeParameters = Types<IFormatProvider>.Array;
110
                if (targetType == typeof(bool))
111
                {
112
                     return convertibleType.GetMethod(nameof(IConvertible.ToBoolean), typeParameters);
113
                }
                else if (targetType == typeof(byte))
115
116
                     return convertibleType.GetMethod(nameof(IConvertible.ToByte), typeParameters);
117
                else if (targetType == typeof(char))
119
                ₹
120
                     return convertibleType.GetMethod(nameof(IConvertible.ToChar), typeParameters);
                }
122
                else if (targetType == typeof(DateTime))
123
124
                     return convertibleType.GetMethod(nameof(IConvertible.ToDateTime),
125

→ typeParameters);

                }
126
                else if (targetType == typeof(decimal))
127
                     return convertibleType.GetMethod(nameof(IConvertible.ToDecimal), typeParameters);
129
130
                else if (targetType == typeof(double))
131
                     return convertibleType.GetMethod(nameof(IConvertible.ToDouble), typeParameters);
133
134
                else if (targetType == typeof(short))
135
136
```

```
return convertibleType.GetMethod(nameof(IConvertible.ToInt16), typeParameters);
137
                 }
                 else if (targetType == typeof(int))
139
140
                     return convertibleType.GetMethod(nameof(IConvertible.ToInt32), typeParameters);
                 }
142
                 else if (targetType == typeof(long))
143
144
                     return convertibleType.GetMethod(nameof(IConvertible.ToInt64), typeParameters);
                 }
146
                 else if (targetType == typeof(sbyte))
147
148
                     return convertibleType.GetMethod(nameof(IConvertible.ToSByte), typeParameters);
149
                 }
150
                 else if (targetType == typeof(float))
151
                     return convertibleType.GetMethod(nameof(IConvertible.ToSingle), typeParameters);
153
154
                 else if (targetType == typeof(string))
155
156
                     return convertibleType.GetMethod(nameof(IConvertible.ToString), typeParameters);
157
158
                 else if (targetType == typeof(ushort))
160
                     return convertibleType.GetMethod(nameof(IConvertible.ToUInt16), typeParameters);
161
                 else if (targetType == typeof(uint))
163
164
                     return convertibleType.GetMethod(nameof(IConvertible.ToUInt32), typeParameters);
165
                 }
                 else if (targetType == typeof(ulong))
167
168
                     return convertibleType.GetMethod(nameof(IConvertible.ToUInt64), typeParameters);
169
                 }
170
                 else
171
                 {
172
                     throw new NotSupportedException();
173
                 }
174
            }
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
                исходного типа (TSource).</para></param>
             /// <returns><para>The value is converted to the target type
182
                 (TTarget).</para><pаra>Значение ковертированное в целевой тип
                 (TTarget).</para></returns>
             [MethodImpl(MethodImplOptions.AggressiveInlining)]
183
            protected static void LoadDefault(ILGenerator il, Type targetType)
184
                 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
191
                     il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(DateTime.MinValue),
192
                     → BindingFlags.Static | BindingFlags.Public));
                 }
193
                 else if (targetType == typeof(decimal))
194
195
                     il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(decimal.Zero),
                        BindingFlags.Static | BindingFlags.Public));
197
                 else if (targetType == typeof(float))
198
199
                     il.LoadConstant(0.0F);
200
201
                 else if (targetType == typeof(double))
202
                     il.LoadConstant(0.0D);
204
205
                 else if (targetType == typeof(long) || targetType == typeof(ulong))
```

```
{
207
                     il.LoadConstant(OL);
                }
20.9
                else
210
                {
211
                     il.LoadConstant(0);
212
                }
213
            }
        }
215
216
1.4
     ./csharp/Platform.Converters/IConverter[TSource, TTarget].cs
   namespace Platform.Converters
 1
        /// <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
 5
            <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
                <typeparamref name="TTarget"/>.</para>
            /// </summary>
            /// <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"/>
16
                type.</para><para>Значение конвертированное в тип <typeparamref

→ name="TTarget"/>.</para></returns>

            TTarget Convert(TSource source);
        }
18
19
     ./csharp/Platform.Converters/IConverter[T].cs
1.5
    namespace Platform.Converters
 1
 2
        /// <summary>
 3
        /// <para>Defines a converter between two values of the same <typeparamref name="T"/>
 4
            type.</para>
        /// <para>Oпределяет 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>
10
    }
11
     ./csharp/Platform. Converters/Unchecked Converter.cs\\
1.6
    using System;
    using System.Runtime.CompilerServices;
 2
    using Platform.Reflection;
 3
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
 6
    namespace Platform.Converters
 7
 8
        /// <summary>
 9
        /// <para>
10
        /// Represents the unchecked converter.
11
        /// </para>
12
        /// <para></para>
13
        /// </summary>
14
        /// <seealso cref="ConverterBase{TSource, TTarget}"/>
15
        public abstract class UncheckedConverter<TSource, TTarget> : ConverterBase<TSource, TTarget>
16
            /// <summary>
18
            /// <para>
19
```

```
/// Gets the default value.
20
            /// </para>
21
            /// <para></para>
22
            /// </summary>
23
            public static UncheckedConverter<TSource, TTarget> Default
24
25
                 [MethodImpl(MethodImplOptions.AggressiveInlining)]
26
27
            } = CompileUncheckedConverter();
28
29
            /// <summary>
30
            /// <para>
31
            /// Compiles the unchecked converter.
32
            /// </para>
33
            /// <para></para>
34
            /// </summary>
            /// <returns>
36
            /// <para>An unchecked converter of t source and t target</para>
37
            /// <para></para>
38
            /// </returns>
39
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
40
            private static UncheckedConverter<TSource, TTarget> CompileUncheckedConverter()
41
                var type = CreateTypeInheritedFrom<UncheckedConverter<TSource, TTarget>>();
43
                EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>());
44
                return (UncheckedConverter<TSource,
45
                    TTarget>) Activator.CreateInstance(type.CreateTypeInfo());
            }
        }
47
   }
48
1.7
     ./csharp/Platform.Converters/UncheckedSignExtendingConverter.cs
   using System;
   using System.Runtime.CompilerServices;
2
   using Platform. Reflection;
3
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
   namespace Platform.Converters
7
        /// <summary>
q
        /// <para>
10
        /// Represents the unchecked sign extending converter.
12
        /// </para>
        /// <para></para>
13
        /// </summary>
14
        /// <seealso cref="ConverterBase{TSource, TTarget}"/>
15
        public abstract class UncheckedSignExtendingConverter<TSource, TTarget> :
16
           ConverterBase<TSource, TTarget>
17
            /// <summary>
18
            /// <para>
19
            /// Gets the default value.
20
            /// </para>
2.1
            /// <para></para>
22
            /// </summary>
            public static UncheckedSignExtendingConverter<TSource, TTarget> Default
24
                 [MethodImpl(MethodImplOptions.AggressiveInlining)]
26
                get:
            } = CompileUncheckedConverter();
29
            /// <summary>
30
            /// <para>
31
            /// 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()
            \hookrightarrow
            {
42
                var type = CreateTypeInheritedFrom<UncheckedSignExtendingConverter<TSource,</pre>
43
                    TTarget>>();
```

```
EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>(extendSign:
44

    true));
                 return (UncheckedSignExtendingConverter<TSource,
45
                    TTarget>) Activator.CreateInstance(type.CreateTypeInfo());
            }
        }
47
   }
48
     ./csharp/Platform.Converters.Tests/ConverterTests.cs
1.8
   using System;
1
   using Xunit;
3
   namespace Platform.Converters.Tests
4
5
        /// <summary>
        /// <para>
        \ensuremath{///} Represents the converter tests.
9
        /// </para>
        /// <para></para>
10
        /// </summary>
11
        public static class ConverterTests
12
13
            /// <summary>
14
            /// <para>
15
            ^{\prime\prime}/// 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.
31
            /// </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>
48
            /// </summary>
49
            [Fact]
50
            public static void SignExtensionTest()
                 var result = UncheckedSignExtendingConverter<byte, long>.Default.Convert(128);
53
                 Assert.Equal(-128L, result);
54
                 result = UncheckedConverter<byte, long>.Default.Convert(128);
55
                 Assert.Equal(128L, result);
56
            }
57
            /// <summary>
59
            /// <para>
60
            /// Tests that object test.
            /// </para>
62
            /// <para></para>
63
            /// </summary>
64
            [Fact]
            public static void ObjectTest()
66
67
                 TestObjectConversion("1");
69
                 TestObjectConversion(DateTime.UtcNow);
```

```
TestObjectConversion(1.0F);
70
                  TestObjectConversion(1.0D);
                  TestObjectConversion(1.0M);
72
                  TestObjectConversion(1UL);
7.3
                  TestObjectConversion(1L);
                  TestObjectConversion(1U);
                  TestObjectConversion(1);
76
                  TestObjectConversion((char)1);
77
                  TestObjectConversion((ushort)1);
                  TestObjectConversion((short)1);
79
                  TestObjectConversion((byte)1);
80
                  TestObjectConversion((sbyte)1);
81
                  TestObjectConversion(true);
             }
83
             /// <summary>
85
             /// <para>
86
             /// Tests the object conversion using the specified value.
             /// </para>
             /// <para></para>
/// </summary>
89
90
             /// <typeparam name="T">
91
             /// <para>The .</para>
92
             /// <para></para>
93
             /// </typeparam>
             /// <param name="value">
/// <para>The value.</para>
/// <para></para>
96
97
             /// </param>
             private static void TestObjectConversion<T>(T value) => Assert.Equal(value,
99

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

         }
100
    }
101
```

## Index

```
./csharp/Platform.Converters.Tests/ConverterTests.cs, 8
./csharp/Platform.Converters/CachingConverterDecorator.cs, 1
./csharp/Platform.Converters/CheckedConverter.cs, 2
./csharp/Platform.Converters/ConverterBase.cs, 2
./csharp/Platform.Converters/IConverter[TSource, TTarget].cs, 6
./csharp/Platform.Converters/IConverter[T].cs, 6
./csharp/Platform.Converters/UncheckedConverter.cs, 6
./csharp/Platform.Converters/UncheckedSignExtendingConverter.cs, 7
```