

## LinksPlatform's Platform.Converters Class Library

### 1.1 ./csharp/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     /// <summary>
10    /// <para>
11    /// Represents the caching converter decorator.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="IConverter{TSource, TTarget}"/>
16    public class CachingConverterDecorator<TSource, TTarget> : IConverter<TSource, TTarget>
17    {
18        private readonly IConverter<TSource, TTarget> _baseConverter;
19        private readonly IDictionary<TSource, TTarget> _cache;
20
21        /// <summary>
22        /// <para>
23        /// Initializes a new <see cref="CachingConverterDecorator"/> instance.
24        /// </para>
25        /// <para></para>
26        /// </summary>
27        /// <param name="baseConverter">
28        /// <para>A base converter.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="cache">
32        /// <para>A cache.</para>
33        /// <para></para>
34        /// </param>
35        [MethodImpl(MethodImplOptions.AggressiveInlining)]
36        public CachingConverterDecorator(IConverter<TSource, TTarget> baseConverter,
37            ↪ IDictionary<TSource, TTarget> cache) => (_baseConverter, _cache) = (baseConverter,
38            ↪ cache);
39
40        /// <summary>
41        /// <para>
42        /// Initializes a new <see cref="CachingConverterDecorator"/> instance.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="baseConverter">
47        /// <para>A base converter.</para>
48        /// <para></para>
49        /// </param>
50        [MethodImpl(MethodImplOptions.AggressiveInlining)]
51        public CachingConverterDecorator(IConverter<TSource, TTarget> baseConverter) :
52            ↪ this(baseConverter, new Dictionary<TSource, TTarget>()) { }
53
54        /// <summary>
55        /// <para>
56        /// Converts the source.
57        /// </para>
58        /// <para></para>
59        /// </summary>
60        /// <param name="source">
61        /// <para>The source.</para>
62        /// <para></para>
63        /// </param>
64        /// <returns>
65        /// <para>The target</para>
66        /// <para></para>
67        /// </returns>
68        [MethodImpl(MethodImplOptions.AggressiveInlining)]
69        public TTarget Convert(TSource source) => _cache.GetOrAdd(source,
70            ↪ _baseConverter.Convert);
71    }
72 }
```

### 1.2 ./csharp/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     /// <summary>
10    /// <para>
11    /// Represents the checked converter.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ConverterBase{TSource, TTarget}"/>
16    public abstract class CheckedConverter<TSource, TTarget> : ConverterBase<TSource, TTarget>
17    {
18        /// <summary>
19        /// <para>
20        /// Gets the default value.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        public static CheckedConverter<TSource, TTarget> Default
25        {
26            [MethodImpl(MethodImplOptions.AggressiveInlining)]
27            get;
28            } = CompileCheckedConverter();
29
30        /// <summary>
31        /// <para>
32        /// Compiles the checked converter.
33        /// </para>
34        /// <para></para>
35        /// </summary>
36        /// <returns>
37        /// <para>A checked converter of t source and t target</para>
38        /// <para></para>
39        /// </returns>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]
41        private static CheckedConverter<TSource, TTarget> CompileCheckedConverter()
42        {
43            var type = CreateTypeInheritedFrom<CheckedConverter<TSource, TTarget>>();
44            EmitConvertMethod(type, il => il.CheckedConvert<TSource, TTarget>());
45            return (CheckedConverter<TSource,
46                ↪ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
47        }
48    }

```

### 1.3 ./csharp/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     /// <summary>
12     /// <para>Represents a base implementation for IConverter interface with the basic logic
13     ↪ necessary for value converter from the <typeparamref name="TSource"/> type to the
14     ↪ <typeparamref name="TTarget"/> type.</para>
15     /// <para>Представляет базовую реализацию для интерфейса IConverter с основной логикой
16     ↪ необходимой для конвертера значений из типа <typeparamref name="TSource"/> в тип
17     ↪ <typeparamref name="TTarget"/>.</para>
18     /// </summary>
19     /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
20     ↪ конверсии.</para></typeparam>
21     /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
22     ↪ конверсии.</para></typeparam>
23     public abstract class ConverterBase<TSource, TTarget> : IConverter<TSource, TTarget>
24     {
25         /// <summary>
26         /// <para>Converts the value of the <typeparamref name="TSource"/> type to the value of
27         ↪ the <typeparamref name="TTarget"/> type.</para>
28         /// <para>Конвертирует значение типа <typeparamref name="TSource"/> в значение типа
29         ↪ <typeparamref name="TTarget"/>.</para>

```

```

22 /// </summary>
23 /// <param name="source"><para>The <typeparamref name="TSource"/> type
    ↳ value.</para><para>Значение типа <typeparamref name="TSource"/>.</para></param>
24 /// <returns><para>The converted value of the <typeparamref name="TTarget"/>
    ↳ type.</para><para>Значение конвертированное в тип <typeparamref
    ↳ name="TTarget"/>.</para></returns>
25 [MethodImpl(MethodImplOptions.AggressiveInlining)]
26 public abstract TTarget Convert(TSource source);
27
28 /// <summary>
29 /// <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>
30 /// <para>Генерирует последовательность инструкций при помощи <see cref="ILGenerator"/>
    ↳ выполняющую преобразование значения типа <see cref="System.Object"/> к значению типа
    ↳ <typeparamref name="TTarget"/>.</para>
31 /// </summary>
32 /// <param name="il"><para>An <see cref="ILGenerator"/> instance.</para><para>Экземпляр
    ↳ <see cref="ILGenerator"/>.</para></param>
33 [MethodImpl(MethodImplOptions.AggressiveInlining)]
34 protected static void ConvertFromObject(ILGenerator il)
35 {
36     var returnDefault = il.DefineLabel();
37     il.Emit(OpCodes.Brfalse_S, returnDefault);
38     il.LoadArgument(1);
39     il.Emit(OpCodes.Castclass, typeof(IConvertible));
40     il.Emit(OpCodes.Ldnull);
41     il.Emit(OpCodes.Callvirt, GetMethodForConversionToTargetType());
42     il.Return();
43     il.MarkLabel(returnDefault);
44     LoadDefault(il, typeof(TTarget));
45 }
46
47 /// <summary>
48 /// <para>Gets a new unique name of an assembly.</para>
49 /// <para>Возвращает новое уникальное имя сборки.</para>
50 /// </summary>
51 /// <returns><para>A new unique name of an assembly.</para><para>Новое уникальное имя
    ↳ сборки.</para></returns>
52 [MethodImpl(MethodImplOptions.AggressiveInlining)]
53 protected static string GetNewName() => Guid.NewGuid().ToString("N");
54
55 /// <summary>
56 /// <para>Converts the value of the source type (TSource) to the value of the target
    ↳ type.</para>
57 /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
58 /// </summary>
59 /// <param name="source"><para>The source type value (TSource).</para><para>Значение
    ↳ исходного типа (TSource).</para></param>
60 /// <returns><para>The value is converted to the target type
    ↳ (TTarget).</para><para>Значение ковертированное в целевой тип
    ↳ (TTarget).</para></returns>
61 [MethodImpl(MethodImplOptions.AggressiveInlining)]
62 protected static TypeBuilder CreateTypeInheritedFrom<TBaseClass>()
63 {
64     var assemblyName = new AssemblyName(GetNewName());
65     var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,
    ↳ AssemblyBuilderAccess.Run);
66     var module = assembly.DefineDynamicModule(GetNewName());
67     var type = module.DefineType(GetNewName(), TypeAttributes.Public |
    ↳ TypeAttributes.Class | TypeAttributes.Sealed, typeof(TBaseClass));
68     return type;
69 }
70
71 /// <summary>
72 /// <para>Converts the value of the source type (TSource) to the value of the target
    ↳ type.</para>
73 /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
74 /// </summary>
75 /// <param name="source"><para>The source type value (TSource).</para><para>Значение
    ↳ исходного типа (TSource).</para></param>
76 /// <returns><para>The value is converted to the target type
    ↳ (TTarget).</para><para>Значение ковертированное в целевой тип
    ↳ (TTarget).</para></returns>
77 [MethodImpl(MethodImplOptions.AggressiveInlining)]
78 protected static void EmitConvertMethod(TypeBuilder typeBuilder, Action<ILGenerator>
    ↳ emitConversion)

```

```

79 {
80     typeBuilder.EmitFinalVirtualMethod<Converter<TSource,
      ↪ TTarget>>(nameof(IConverter<TSource, TTarget>.Convert), il =>
81     {
82         il.LoadArgument(1);
83         if (typeof(TSource) == typeof(object) && typeof(TTarget) != typeof(object))
84         {
85             ConvertFromObject(il);
86         }
87         else if (typeof(TSource) != typeof(object) && typeof(TTarget) == typeof(object))
88         {
89             il.Box(typeof(TSource));
90         }
91         else
92         {
93             emitConversion(il);
94         }
95         il.Return();
96     });
97 }
98
99 /// <summary>
100 /// <para>Converts the value of the source type (TSource) to the value of the target
      ↪ type.</para>
101 /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
102 /// </summary>
103 /// <param name="source"><para>The source type value (TSource).</para><para>Значение
      ↪ исходного типа (TSource).</para></param>
104 /// <returns><para>The value is converted to the target type
      ↪ (TTarget).</para><para>Значение ковертированное в целевой тип
      ↪ (TTarget).</para></returns>
105 [MethodImpl(MethodImplOptions.AggressiveInlining)]
106 protected static MethodInfo GetMethodForConversionToTargetType()
107 {
108     var targetType = typeof(TTarget);
109     var convertibleType = typeof(IConvertible);
110     var typeParameters = Types<IFormatProvider>.Array;
111     if (targetType == typeof(bool))
112     {
113         return convertibleType.GetMethod(nameof(IConvertible.ToBoolean), typeParameters);
114     }
115     else if (targetType == typeof(byte))
116     {
117         return convertibleType.GetMethod(nameof(IConvertible.ToByte), typeParameters);
118     }
119     else if (targetType == typeof(char))
120     {
121         return convertibleType.GetMethod(nameof(IConvertible.ToChar), typeParameters);
122     }
123     else if (targetType == typeof(DateTime))
124     {
125         return convertibleType.GetMethod(nameof(IConvertible.ToDateTime),
      ↪ typeParameters);
126     }
127     else if (targetType == typeof(decimal))
128     {
129         return convertibleType.GetMethod(nameof(IConvertible.ToDecimal), typeParameters);
130     }
131     else if (targetType == typeof(double))
132     {
133         return convertibleType.GetMethod(nameof(IConvertible.ToDouble), typeParameters);
134     }
135     else if (targetType == typeof(short))
136     {
137         return convertibleType.GetMethod(nameof(IConvertible.ToInt16), typeParameters);
138     }
139     else if (targetType == typeof(int))
140     {
141         return convertibleType.GetMethod(nameof(IConvertible.ToInt32), typeParameters);
142     }
143     else if (targetType == typeof(long))
144     {
145         return convertibleType.GetMethod(nameof(IConvertible.ToInt64), typeParameters);
146     }
147     else if (targetType == typeof(sbyte))
148     {
149         return convertibleType.GetMethod(nameof(IConvertible.ToSByte), typeParameters);

```

```

150     }
151     else if (targetType == typeof(float))
152     {
153         return convertibleType.GetMethod(nameof(Convertible.ToSingle), typeParameters);
154     }
155     else if (targetType == typeof(string))
156     {
157         return convertibleType.GetMethod(nameof(Convertible.ToString), typeParameters);
158     }
159     else if (targetType == typeof(ushort))
160     {
161         return convertibleType.GetMethod(nameof(Convertible.ToUInt16), typeParameters);
162     }
163     else if (targetType == typeof(uint))
164     {
165         return convertibleType.GetMethod(nameof(Convertible.ToUInt32), typeParameters);
166     }
167     else if (targetType == typeof(ulong))
168     {
169         return convertibleType.GetMethod(nameof(Convertible.ToUInt64), typeParameters);
170     }
171     else
172     {
173         throw new NotSupportedException();
174     }
175 }
176
177 /// <summary>
178 /// <para>Converts the value of the source type (TSource) to the value of the target
179   ↳ type.</para>
180 /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
181 /// </summary>
182 /// <param name="source"><para>The source type value (TSource).</para><para>Значение
   ↳ исходного типа (TSource).</para></param>
183 /// <returns><para>The value is converted to the target type
   ↳ (TTarget).</para><para>Значение конвертированное в целевой тип
   ↳ (TTarget).</para></returns>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 protected static void LoadDefault(ILGenerator il, Type targetType)
186 {
187     if (targetType == typeof(string))
188     {
189         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(string.Empty),
190   ↳ BindingFlags.Static | BindingFlags.Public));
191     }
192     else if (targetType == typeof(DateTime))
193     {
194         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(DateTime.MinValue),
195   ↳ BindingFlags.Static | BindingFlags.Public));
196     }
197     else if (targetType == typeof(decimal))
198     {
199         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(decimal.Zero),
200   ↳ BindingFlags.Static | BindingFlags.Public));
201     }
202     else if (targetType == typeof(float))
203     {
204         il.LoadConstant(0.0F);
205     }
206     else if (targetType == typeof(double))
207     {
208         il.LoadConstant(0.0D);
209     }
210     else if (targetType == typeof(long) || targetType == typeof(ulong))
211     {
212         il.LoadConstant(0L);
213     }
214     else
215     {
216         il.LoadConstant(0);
217     }
218 }
219 }
220 }
221 }
222 }

```

#### 1.4 ./csharp/Platform.Converters/IConverter[TSource, TTarget].cs

```
1 namespace Platform.Converters
2 {
3     /// <summary>
4     /// <para>Defines a value converter from the <typeparamref name="TSource"/> type to the
5     /// <para><typeparamref name="TTarget"/> type.</para>
6     /// <para>Определяет конвертер значений из типа <typeparamref name="TSource"/> в тип
7     /// <para><typeparamref name="TTarget"/>.</para>
8     /// </summary>
9     /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
10    /// <para>конверсии.</para></typeparam>
11    /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
12    /// <para>конверсии.</para></typeparam>
13    public interface IConverter<in TSource, out TTarget>
14    {
15        /// <summary>
16        /// <para>Converts the value of the <typeparamref name="TSource"/> type to the value of
17        /// <para>the <typeparamref name="TTarget"/> type.</para>
18        /// <para>Конвертирует значение типа <typeparamref name="TSource"/> в значение типа
19        /// <para><typeparamref name="TTarget"/>.</para>
20        /// </summary>
21        /// <param name="source"><para>The <typeparamref name="TSource"/> type
22        /// <para>value.</para><para>Значение типа <typeparamref name="TSource"/>.</para></param>
23        /// <returns><para>The converted value of the <typeparamref name="TTarget"/>
24        /// <para>type.</para><para>Значение конвертированное в тип <typeparamref
25        /// <para>name="TTarget"/>.</para></returns>
26        TTarget Convert(TSource source);
27    }
28 }
```

#### 1.5 ./csharp/Platform.Converters/IConverter[T].cs

```
1 namespace Platform.Converters
2 {
3     /// <summary>
4     /// <para>Defines a converter between two values of the same <typeparamref name="T"/>
5     /// <para>type.</para>
6     /// <para>Определяет конвертер между двумя значениями одного типа <typeparamref
7     /// <para>name="T"/>.</para>
8     /// </summary>
9     /// <typeparam name="T"><para>The type of value to convert.</para><para>Тип преобразуемого
10    /// <para>значения.</para></typeparam>
11    public interface IConverter<T> : IConverter<T, T>
12    {
13    }
14 }
```

#### 1.6 ./csharp/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     /// <summary>
10    /// <para>
11    /// <para>Represents the unchecked converter.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ConverterBase{TSource, TTarget}">
16    public abstract class UncheckedConverter<TSource, TTarget> : ConverterBase<TSource, TTarget>
17    {
18        /// <summary>
19        /// <para>
20        /// <para>Gets the default value.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        public static UncheckedConverter<TSource, TTarget> Default
25        {
26            [MethodImpl(MethodImplOptions.AggressiveInlining)]
27            get;
28        } = CompileUncheckedConverter();
29
30        /// <summary>
```

```

31     /// <para>
32     /// Compiles the unchecked converter.
33     /// </para>
34     /// <para></para>
35     /// </summary>
36     /// <returns>
37     /// <para>An unchecked converter of t source and t target</para>
38     /// <para></para>
39     /// </returns>
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     private static UncheckedConverter<TSource, TTarget> CompileUncheckedConverter()
42     {
43         var type = CreateTypeInheritedFrom<UncheckedConverter<TSource, TTarget>>();
44         EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>());
45         return (UncheckedConverter<TSource,
46             ↪ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
47     }
48 }

```

## 1.7 ./csharp/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      /// <summary>
10     /// <para>
11     /// Represents the unchecked sign extending converter.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ConverterBase{TSource, TTarget}"/>
16     public abstract class UncheckedSignExtendingConverter<TSource, TTarget> :
17     ↪ ConverterBase<TSource, TTarget>
18     {
19         /// <summary>
20         /// <para>
21         /// Gets the default value.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public static UncheckedSignExtendingConverter<TSource, TTarget> Default
26         {
27             [MethodImpl(MethodImplOptions.AggressiveInlining)]
28             get;
29         } = CompileUncheckedConverter();
30
31         /// <summary>
32         /// <para>
33         /// Compiles the unchecked converter.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         /// <returns>
38         /// <para>An unchecked sign extending converter of t source and t target</para>
39         /// <para></para>
40         /// </returns>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         private static UncheckedSignExtendingConverter<TSource, TTarget>
43         ↪ CompileUncheckedConverter()
44         {
45             var type = CreateTypeInheritedFrom<UncheckedSignExtendingConverter<TSource,
46                 ↪ TTarget>>();
47             EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>(extendSign:
48                 ↪ true));
49             return (UncheckedSignExtendingConverter<TSource,
50                 ↪ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
51         }
52     }
53 }

```

## 1.8 ./csharp/Platform.Converters.Tests/ConverterTests.cs

```

1  using System;
2  using Xunit;

```

```

3
4 namespace Platform.Converters.Tests
5 {
6     /// <summary>
7     /// <para>
8     /// Represents the converter tests.
9     /// </para>
10    /// <para></para>
11    /// </summary>
12    public static class ConverterTests
13    {
14        /// <summary>
15        /// <para>
16        /// Tests that same type test.
17        /// </para>
18        /// <para></para>
19        /// </summary>
20        [Fact]
21        public static void SameTypeTest()
22        {
23            var result = UncheckedConverter<ulong, ulong>.Default.Convert(2UL);
24            Assert.Equal(2UL, result);
25            result = CheckedConverter<ulong, ulong>.Default.Convert(2UL);
26            Assert.Equal(2UL, result);
27        }
28
29        /// <summary>
30        /// <para>
31        /// Tests that int 32 to u int 64 test.
32        /// </para>
33        /// <para></para>
34        /// </summary>
35        [Fact]
36        public static void Int32ToUInt64Test()
37        {
38            var result = UncheckedConverter<int, ulong>.Default.Convert(2);
39            Assert.Equal(2UL, result);
40            result = CheckedConverter<int, ulong>.Default.Convert(2);
41            Assert.Equal(2UL, result);
42        }
43
44        /// <summary>
45        /// <para>
46        /// Tests that sign extension test.
47        /// </para>
48        /// <para></para>
49        /// </summary>
50        [Fact]
51        public static void SignExtensionTest()
52        {
53            var result = UncheckedSignExtendingConverter<byte, long>.Default.Convert(128);
54            Assert.Equal(-128L, result);
55            result = UncheckedConverter<byte, long>.Default.Convert(128);
56            Assert.Equal(128L, result);
57        }
58
59        /// <summary>
60        /// <para>
61        /// Tests that object test.
62        /// </para>
63        /// <para></para>
64        /// </summary>
65        [Fact]
66        public static void ObjectTest()
67        {
68            TestObjectConversion("1");
69            TestObjectConversion(DateTime.UtcNow);
70            TestObjectConversion(1.0F);
71            TestObjectConversion(1.0D);
72            TestObjectConversion(1.0M);
73            TestObjectConversion(1UL);
74            TestObjectConversion(1L);
75            TestObjectConversion(1U);
76            TestObjectConversion(1);
77            TestObjectConversion((char)1);
78            TestObjectConversion((ushort)1);
79            TestObjectConversion((short)1);
80            TestObjectConversion((byte)1);

```



```
81         TestObjectConversion((sbyte)1);
82         TestObjectConversion(true);
83     }
84
85     private static void TestObjectConversion<T>(T value) => Assert.Equal(value,
86         ↪ UncheckedConverter<object, T>.Default.Convert(value));
87 }
```

## 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