

## 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     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 ./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     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 ./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>Defines a converter between two types (TSource and TTarget).</para>
13     /// <para>Определяет конвертер между двумя типами (исходным TSource и целевым
14     ↪ TTarget).</para>
15     /// </summary>
16     /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
17     ↪ конверсии.</para></typeparam>
18     /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
19     ↪ конверсии.</para></typeparam>
20     public abstract class ConverterBase<TSource, TTarget> : IConverter<TSource, TTarget>
```

```

18 {
19     /// <summary>
20     /// <para>Converts the value of the source type (TSource) to the value of the target
    ↳ type.</para>
21     /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
22     /// </summary>
23     /// <param name="source"><para>The source type value (TSource).</para><para>Значение
    ↳ исходного типа (TSource).</para></param>
24     /// <returns><para>The value is converted to the target type
    ↳ (TTarget).</para><para>Значение ковертированное в целевой тип
    ↳ (TTarget).</para></returns>
25     [MethodImpl(MethodImplOptions.AggressiveInlining)]
26     public abstract TTarget Convert(TSource source);
27
28     /// <summary>
29     /// <para>Converts the value of the source type (TSource) to the value of the target
    ↳ type.</para>
30     /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
31     /// </summary>
32     /// <param name="source"><para>The source type value (TSource).</para><para>Значение
    ↳ исходного типа (TSource).</para></param>
33     /// <returns><para>The value is converted to the target type
    ↳ (TTarget).</para><para>Значение ковертированное в целевой тип
    ↳ (TTarget).</para></returns>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     protected static void ConvertFromObject(ILGenerator il)
36     {
37         var returnDefault = il.DefineLabel();
38         il.Emit(OpCodes.Brfalse_S, returnDefault);
39         il.LoadArgument(1);
40         il.Emit(OpCodes.Castclass, typeof(IConvertible));
41         il.Emit(OpCodes.Ldnull);
42         il.Emit(OpCodes.Callvirt, GetMethodForConversionToTargetType());
43         il.Return();
44         il.MarkLabel(returnDefault);
45         LoadDefault(il, typeof(TTarget));
46     }
47
48     /// <summary>
49     /// <para>Converts the value of the source type (TSource) to the value of the target
    ↳ type.</para>
50     /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
51     /// </summary>
52     /// <param name="source"><para>The source type value (TSource).</para><para>Значение
    ↳ исходного типа (TSource).</para></param>
53     /// <returns><para>The value is converted to the target type
    ↳ (TTarget).</para><para>Значение ковертированное в целевой тип
    ↳ (TTarget).</para></returns>
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     protected static string GetNewName() => Guid.NewGuid().ToString("N");
56
57     /// <summary>
58     /// <para>Converts the value of the source type (TSource) to the value of the target
    ↳ type.</para>
59     /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
60     /// </summary>
61     /// <param name="source"><para>The source type value (TSource).</para><para>Значение
    ↳ исходного типа (TSource).</para></param>
62     /// <returns><para>The value is converted to the target type
    ↳ (TTarget).</para><para>Значение ковертированное в целевой тип
    ↳ (TTarget).</para></returns>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected static TypeBuilder CreateTypeInheritedFrom<TBaseClass>()
65     {
66         var assemblyName = new AssemblyName(GetNewName());
67         var assembly = AssemblyBuilder.DefineDynamicAssembly(assemblyName,
    ↳ AssemblyBuilderAccess.Run);
68         var module = assembly.DefineDynamicModule(GetNewName());
69         var type = module.DefineType(GetNewName(), TypeAttributes.Public |
    ↳ TypeAttributes.Class | TypeAttributes.Sealed, typeof(TBaseClass));
70         return type;
71     }
72
73     /// <summary>
74     /// <para>Converts the value of the source type (TSource) to the value of the target
    ↳ type.</para>

```

```

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

```

```

142     {
143         return convertibleType.GetMethod(nameof(IConvertible.ToInt32), typeParameters);
144     }
145     else if (targetType == typeof(long))
146     {
147         return convertibleType.GetMethod(nameof(IConvertible.ToInt64), typeParameters);
148     }
149     else if (targetType == typeof(sbyte))
150     {
151         return convertibleType.GetMethod(nameof(IConvertible.ToSByte), typeParameters);
152     }
153     else if (targetType == typeof(float))
154     {
155         return convertibleType.GetMethod(nameof(IConvertible.ToSingle), typeParameters);
156     }
157     else if (targetType == typeof(string))
158     {
159         return convertibleType.GetMethod(nameof(IConvertible.ToString), typeParameters);
160     }
161     else if (targetType == typeof(ushort))
162     {
163         return convertibleType.GetMethod(nameof(IConvertible.ToUInt16), typeParameters);
164     }
165     else if (targetType == typeof(uint))
166     {
167         return convertibleType.GetMethod(nameof(IConvertible.ToUInt32), typeParameters);
168     }
169     else if (targetType == typeof(ulong))
170     {
171         return convertibleType.GetMethod(nameof(IConvertible.ToUInt64), typeParameters);
172     }
173     else
174     {
175         throw new NotSupportedException();
176     }
177 }
178
179 /// <summary>
180 /// <para>Converts the value of the source type (TSource) to the value of the target
181   ↳ type.</para>
182 /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
183 /// </summary>
184 /// <param name="source"><para>The source type value (TSource).</para><para>Значение
185   ↳ исходного типа (TSource).</para></param>
186 /// <returns><para>The value is converted to the target type
187   ↳ (TTarget).</para><para>Значение ковертированное в целевой тип
188   ↳ (TTarget).</para></returns>
189 [MethodImpl(MethodImplOptions.AggressiveInlining)]
190 protected static void LoadDefault(ILGenerator il, Type targetType)
191 {
192     if (targetType == typeof(string))
193     {
194         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(string.Empty),
195             ↳ BindingFlags.Static | BindingFlags.Public));
196     }
197     else if (targetType == typeof(DateTime))
198     {
199         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(DateTime.MinValue),
200             ↳ BindingFlags.Static | BindingFlags.Public));
201     }
202     else if (targetType == typeof(decimal))
203     {
204         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(decimal.Zero),
205             ↳ BindingFlags.Static | BindingFlags.Public));
206     }
207     else if (targetType == typeof(float))
208     {
209         il.LoadConstant(0.0F);
210     }
211     else if (targetType == typeof(double))
212     {
213         il.LoadConstant(0.0D);
214     }
215     else if (targetType == typeof(long) || targetType == typeof(ulong))
216     {
217         il.LoadConstant(0L);
218     }
219 }

```

```

212         else
213         {
214             il.LoadConstant(0);
215         }
216     }
217 }
218 }

```

#### 1.4 ./csharp/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
6     ///     ↪ TTarget).</para>
7     /// </summary>
8     /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
9     ///     ↪ конверсии.</para></typeparam>
10    /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
11    ///     ↪ конверсии.</para></typeparam>
12    public interface IConverter<in TSource, out TTarget>
13    {
14        /// <summary>
15        /// <para>Converts the value of the source type (TSource) to the value of the target
16        ///     ↪ type.</para>
17        /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
18        /// </summary>
19        /// <param name="source"><para>The source type value (TSource).</para><para>Значение
20        ///     ↪ исходного типа (TSource).</para></param>
21        /// <returns><para>The value is converted to the target type
22        ///     ↪ (TTarget).</para><para>Значение ковертированное в целевой тип
23        ///     ↪ (TTarget).</para></returns>
24        TTarget Convert(TSource source);
25    }
26 }

```

#### 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 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
8     ///     ↪ значения.</para></typeparam>
9     public interface IConverter<T> : IConverter<T, T>
10    {
11    }
12 }

```

#### 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     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 = CreateTypeInheritedFrom<UncheckedConverter<TSource, TTarget>>();
21            EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>());
22            return (UncheckedConverter<TSource,
23                ↪ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
24        }
25    }
26 }

```

### 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     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()
21         {
22             var type = CreateTypeInheritedFrom<UncheckedSignExtendingConverter<TSource,
23             ↪ TTarget>>();
24             EmitConvertMethod(type, il => il.UncheckedConvert<TSource, TTarget>(extendSign:
25             ↪ true));
26             return (UncheckedSignExtendingConverter<TSource,
27             ↪ TTarget>)Activator.CreateInstance(type.CreateTypeInfo());
28         }
29     }
30 }
```

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

```
1 using System;
2 using Xunit;
3
4 namespace Platform.Converters.Tests
5 {
6     public static class ConverterTests
7     {
8         [Fact]
9         public static void SameTypeTest()
10         {
11             var result = UncheckedConverter<ulong, ulong>.Default.Convert(2UL);
12             Assert.Equal(2UL, result);
13             result = CheckedConverter<ulong, ulong>.Default.Convert(2UL);
14             Assert.Equal(2UL, result);
15         }
16
17         [Fact]
18         public static void Int32ToUInt64Test()
19         {
20             var result = UncheckedConverter<int, ulong>.Default.Convert(2);
21             Assert.Equal(2UL, result);
22             result = CheckedConverter<int, ulong>.Default.Convert(2);
23             Assert.Equal(2UL, result);
24         }
25
26         [Fact]
27         public static void SignExtensionTest()
28         {
29             var result = UncheckedSignExtendingConverter<byte, long>.Default.Convert(128);
30             Assert.Equal(-128L, result);
31             result = UncheckedConverter<byte, long>.Default.Convert(128);
32             Assert.Equal(128L, result);
33         }
34
35         [Fact]
36         public static void ObjectTest()
37         {
38             TestObjectConversion("1");
39             TestObjectConversion(DateTime.UtcNow);
40             TestObjectConversion(1.0F);
41             TestObjectConversion(1.0D);
42             TestObjectConversion(1.0M);
43             TestObjectConversion(1UL);
44             TestObjectConversion(1L);
45             TestObjectConversion(1U);
46             TestObjectConversion(1);
47         }
48     }
49 }
```

```
47         TestObjectConversion((char)1);
48         TestObjectConversion((ushort)1);
49         TestObjectConversion((short)1);
50         TestObjectConversion((byte)1);
51         TestObjectConversion((sbyte)1);
52         TestObjectConversion(true);
53     }
54
55     private static void TestObjectConversion<T>(T value) => Assert.Equal(value,
56         ↪ UncheckedConverter<object, T>.Default.Convert(value));
57 }
```

## Index

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