

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>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>
```

```

16  /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
    ↳ конверсии.</para></typeparam>
17  public abstract class ConverterBase<TSource, TTarget> : IConverter<TSource, TTarget>
18  {
19      /// <summary>
20      /// <para>Converts the value of the <typeparamref name="TSource"/> type to the value of
    ↳ the <typeparamref name="TTarget"/> type.</para>
21      /// <para>Конвертирует значение типа <typeparamref name="TSource"/> в значение типа
    ↳ <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(IConvertible.ToSingle), typeParameters);
154     }
155     else if (targetType == typeof(string))
156     {
157         return convertibleType.GetMethod(nameof(IConvertible.ToString), typeParameters);
158     }
159     else if (targetType == typeof(ushort))
160     {
161         return convertibleType.GetMethod(nameof(IConvertible.ToUInt16), typeParameters);
162     }
163     else if (targetType == typeof(uint))
164     {
165         return convertibleType.GetMethod(nameof(IConvertible.ToUInt32), typeParameters);
166     }
167     else if (targetType == typeof(ulong))
168     {
169         return convertibleType.GetMethod(nameof(IConvertible.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
183   ↳ исходного типа (TSource).</para></param>
184 /// <returns><para>The value is converted to the target type
185   ↳ (TTarget).</para><para>Значение ковертированное в целевой тип
186   ↳ (TTarget).</para></returns>
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 protected static void LoadDefault(ILGenerator il, Type targetType)
189 {
190     if (targetType == typeof(string))
191     {
192         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(string.Empty),
193   ↳ BindingFlags.Static | BindingFlags.Public));
194     }
195     else if (targetType == typeof(DateTime))
196     {
197         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(DateTime.MinValue),
198   ↳ BindingFlags.Static | BindingFlags.Public));
199     }
200     else if (targetType == typeof(decimal))
201     {
202         il.Emit(OpCodes.Ldsfld, targetType.GetField(nameof(decimal.Zero),
203   ↳ BindingFlags.Static | BindingFlags.Public));
204     }
205     else if (targetType == typeof(float))
206     {
207         il.LoadConstant(0.0F);
208     }
209     else if (targetType == typeof(double))
210     {
211         il.LoadConstant(0.0D);
212     }
213     else if (targetType == typeof(long) || targetType == typeof(ulong))
214     {
215         il.LoadConstant(0L);
216     }
217 }

```

```

210         else
211         {
212             il.LoadConstant(0);
213         }
214     }
215 }
216 }

```

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     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 }

```

```

23     }
24 }
25 }

```

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     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,
    ↪    UncheckedConverter<object, T>.Default.Convert(value));
56 }
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, 6