LinksPlatform's Platform.Reflection.Sigil Class Library

./Platform.Reflection.Sigil/DelegateHelpers.cs

```
using System;
using System.Collections.Generic;
using Sigil;
using Platform.Exceptions;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Reflection.Sigil
{
    public static class DelegateHelpers
    {
        public static TDelegate Compile<TDelegate>(Action<Emit<TDelegate>> emitCode)
        {
            var @delegate = default(TDelegate);
            try
            {
                var emiter = Emit<TDelegate>.NewDynamicMethod();
                emitCode(emiter);
                @delegate = emiter.CreateDelegate();
            }
            catch (Exception exception)
            {
                exception.Ignore();
            }
            finally
            {
                if (EqualityComparer<TDelegate>.Default.Equals(@delegate, default))
                {
                    var factory = new NotSupportedExceptionDelegateFactory<TDelegate>();
                    @delegate = factory.Create();
                }
            }
            return @delegate;
        }
    }
}
```

./Platform.Reflection.Sigil/EmitExtensions.cs

```
using System;
using Sigil;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Reflection.Sigil
{
    public static class EmitExtensions
    {
        public static Emit<TDelegate> LoadConstantOne<TDelegate>(this Emit<TDelegate> emiter,
        ↪  Type constantType)
        {
            if (constantType == typeof(float))
            {
                emiter.LoadConstant(1F);
            }
            else if (constantType == typeof(double))
            {
                emiter.LoadConstant(1D);
            }
            else if (constantType == typeof(long))
            {
                emiter.LoadConstant(1L);
            }
            else if (constantType == typeof(ulong))
            {
                emiter.LoadConstant(1UL);
            }
            else if (constantType == typeof(int))
            {
                emiter.LoadConstant(1);
            }
            else if (constantType == typeof(uint))
            {
                emiter.LoadConstant(1U);
            }
            else if (constantType == typeof(short))
            {
                emiter.LoadConstant(1);
```

```
                emiter.Convert<short>();
            }
            else if (constantType == typeof(ushort))
            {
                emiter.LoadConstant(1);
                emiter.Convert<ushort>();
            }
            else if (constantType == typeof(sbyte))
            {
                emiter.LoadConstant(1);
                emiter.Convert<sbyte>();
            }
            else if (constantType == typeof(byte))
            {
                emiter.LoadConstant(1);
                emiter.Convert<byte>();
            }
            else
            {
                throw new NotSupportedException();
            }
            return emiter;
        }

        public static Emit<TDelegate> LoadConstant<TDelegate>(this Emit<TDelegate> emiter, Type
        ↪  constantType, object constantValue)
        {
            if (constantType == typeof(float))
            {
                emiter.LoadConstant((float)constantValue);
            }
            else if (constantType == typeof(double))
            {
                emiter.LoadConstant((double)constantValue);
            }
            else if (constantType == typeof(long))
            {
                emiter.LoadConstant((long)constantValue);
            }
            else if (constantType == typeof(ulong))
            {
                emiter.LoadConstant((ulong)constantValue);
            }
            else if (constantType == typeof(int))
            {
                emiter.LoadConstant((int)constantValue);
            }
            else if (constantType == typeof(uint))
            {
                emiter.LoadConstant((uint)constantValue);
            }
            else if (constantType == typeof(short))
            {
                emiter.LoadConstant((short)constantValue);
                emiter.Convert<short>();
            }
            else if (constantType == typeof(ushort))
            {
                emiter.LoadConstant((ushort)constantValue);
                emiter.Convert<ushort>();
            }
            else if (constantType == typeof(sbyte))
            {
                emiter.LoadConstant((sbyte)constantValue);
                emiter.Convert<sbyte>();
            }
            else if (constantType == typeof(byte))
            {
                emiter.LoadConstant((byte)constantValue);
                emiter.Convert<byte>();
            }
            else
            {
                throw new NotSupportedException();
            }
            return emiter;
        }

```

```
116         public static Emit<TDelegate> Increment<TDelegate>(this Emit<TDelegate> emiter, Type
            ↪ valueType)
117         {
118             emiter.LoadConstantOne(valueType);
119             emiter.Add();
120             return emiter;
121         }
122 
123         public static Emit<TDelegate> Decrement<TDelegate>(this Emit<TDelegate> emiter, Type
            ↪ valueType)
124         {
125             emiter.LoadConstantOne(valueType);
126             emiter.Subtract();
127             return emiter;
128         }
129 
130         public static Emit<TDelegate> LoadArguments<TDelegate>(this Emit<TDelegate> emiter,
            ↪ params ushort[] arguments)
131         {
132             for (var i = 0; i < arguments.Length; i++)
133             {
134                 emiter.LoadArgument(arguments[i]);
135             }
136             return emiter;
137         }
138 
139         public static Emit<TDelegate> CompareGreaterThan<TDelegate>(this Emit<TDelegate> emiter,
            ↪ bool isSigned)
140         {
141             if (isSigned)
142             {
143                 emiter.CompareGreaterThan();
144             }
145             else
146             {
147                 emiter.UnsignedCompareGreaterThan();
148             }
149             return emiter;
150         }
151 
152         public static Emit<TDelegate> CompareLessThan<TDelegate>(this Emit<TDelegate> emiter,
            ↪ bool isSigned)
153         {
154             if (isSigned)
155             {
156                 emiter.CompareLessThan();
157             }
158             else
159             {
160                 emiter.UnsignedCompareLessThan();
161             }
162             return emiter;
163         }
164 
165         public static Emit<TDelegate> BranchIfGreaterOrEqual<TDelegate>(this Emit<TDelegate>
            ↪ emiter, bool isSigned, Label label)
166         {
167             if (isSigned)
168             {
169                 emiter.BranchIfGreaterOrEqual(label);
170             }
171             else
172             {
173                 emiter.UnsignedBranchIfGreaterOrEqual(label);
174             }
175             return emiter;
176         }
177 
178         public static Emit<TDelegate> BranchIfLessOrEqual<TDelegate>(this Emit<TDelegate>
            ↪ emiter, bool isSigned, Label label)
179         {
180             if (isSigned)
181             {
182                 emiter.BranchIfLessOrEqual(label);
183             }
184             else
185             {
186                 emiter.UnsignedBranchIfLessOrEqual(label);
```

```
            }
            return emiter;
        }
    }
}
```

./Platform.Reflection.Sigil/NotSupportedExceptionDelegateFactory.cs

```
using System;
using Sigil;
using Platform.Interfaces;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Reflection.Sigil
{
    public class NotSupportedExceptionDelegateFactory<TDelegate> : IFactory<TDelegate>
    {
        public TDelegate Create()
        {
            var emiter = Emit<TDelegate>.NewDynamicMethod();
            emiter.NewObject<NotSupportedException>();
            emiter.Throw();
            return emiter.CreateDelegate();
        }
    }
}
```

./Platform.Reflection.Sigil.Tests/InlineTests.cs

```
using System;
using System.Linq;
using System.Reflection;
using System.Reflection.Emit;
using Xunit;
using TheSigil = Sigil;

namespace Platform.Reflection.Sigil.Tests
{
    public static class InlineTests
    {
        [Fact]
        public static void SimpleInlineTest()
        {
            var disassembledOuterMethod = TheSigil.Disassembler<Action>.Disassemble(OuterMethod);
            var emiter = TheSigil.Emit<Action>.NewDynamicMethod();

            foreach (var operation in disassembledOuterMethod)
            {
                if (operation.OpCode == OpCodes.Nop)
                {
                    continue;
                }
                if (operation.OpCode == OpCodes.Call)
                {
                    var firstParameter = operation.Parameters.First();
                    if (firstParameter is MethodInfo methodInfo)
                    {
                        if (methodInfo.Name == nameof(InnerMethod))
                        {
                            var disassembledInnerMethod =
                            ↪  TheSigil.Disassembler<Action>.Disassemble(InnerMethod);
                            var i = 0;
                            foreach (var innerOperation in disassembledInnerMethod)
                            {
                                // There is no way to replay operation at emiter
                                // emiter.Replay(innerOperation)

                                // There is also no way to rewrite operations in the
                                ↪  disassembledOuterMethod
                                // disassembledOuterMethod[i++] = innerOperation;
                            }

                            // So the only way (but it is not practical for now) is to use:
                            emiter = disassembledInnerMethod.EmitAll();
                            // That means we just use InnerMethod method directly,
                            // but if OuterMethod will contain something else we will fail with
                            ↪  current support of disassembling in the Sigil.
                        }
                    }
                }
```

```
            }

            Action result = emiter.CreateDelegate();
            result();
        }

        private static void InnerMethod()
        {
            Console.WriteLine("Inner method.");
        }

        private static void OuterMethod()
        {
            InnerMethod();
        }
    }
}
```

Index