

LinksPlatform's Platform.Reflection.Sigil Class Library

./DelegateHelpers.cs

```
1 using System;
2 using System.Collections.Generic;
3 using Sigil;
4 using Platform.Exceptions;
5
6 namespace Platform.Reflection.Sigil
7 {
8     public static class DelegateHelpers
9     {
10         public static TDelegate Compile<TDelegate>(Action<Emit<TDelegate>> emitCode)
11         {
12             var @delegate = default(TDelegate);
13             try
14             {
15                 var emitter = Emit<TDelegate>.NewDynamicMethod();
16                 emitCode(emitter);
17                 @delegate = emitter.CreateDelegate();
18             }
19             catch (Exception exception)
20             {
21                 exception.Ignore();
22             }
23             finally
24             {
25                 if (EqualityComparer<TDelegate>.Default.Equals(@delegate, default))
26                 {
27                     var factory = new NotSupportedExceptionDelegateFactory<TDelegate>();
28                     @delegate = factory.Create();
29                 }
30             }
31             return @delegate;
32         }
33     }
34 }
```

./EmitExtensions.cs

```
1 using System;
2 using Sigil;
3
4 namespace Platform.Reflection.Sigil
5 {
6     public static class EmitExtensions
7     {
8         public static Emit<TDelegate> LoadConstantOne<TDelegate>(this Emit<TDelegate> emitter,
9             ↪ Type constantType)
10         {
11             if (constantType == typeof(float))
12             {
13                 emitter.LoadConstant(1F);
14             }
15             else if (constantType == typeof(double))
16             {
17                 emitter.LoadConstant(1D);
18             }
19             else if (constantType == typeof(long))
20             {
21                 emitter.LoadConstant(1L);
22             }
23             else if (constantType == typeof(ulong))
24             {
25                 emitter.LoadConstant(1UL);
26             }
27             else if (constantType == typeof(int))
28             {
29                 emitter.LoadConstant(1);
30             }
31             else if (constantType == typeof(uint))
32             {
33                 emitter.LoadConstant(1U);
34             }
35             else if (constantType == typeof(short))
36             {
37                 emitter.LoadConstant(1);
38                 emitter.Convert<short>();
39             }
40             else if (constantType == typeof(ushort))
41             {
42                 emitter.LoadConstant(1);
43                 emitter.Convert<ushort>();
44             }
45         }
46     }
47 }
```

```

41         emitter.LoadConstant(1);
42         emitter.Convert<ushort>();
43     }
44     else if (constantType == typeof(sbyte))
45     {
46         emitter.LoadConstant(1);
47         emitter.Convert<sbyte>();
48     }
49     else if (constantType == typeof(byte))
50     {
51         emitter.LoadConstant(1);
52         emitter.Convert<byte>();
53     }
54     else
55     {
56         throw new NotSupportedException();
57     }
58     return emitter;
59 }
60
61 public static Emit<TDelegate> LoadConstant<TDelegate>(this Emit<TDelegate> emitter, Type
→ constantType, object constantValue)
62 {
63     if (constantType == typeof(float))
64     {
65         emitter.LoadConstant((float)constantValue);
66     }
67     else if (constantType == typeof(double))
68     {
69         emitter.LoadConstant((double)constantValue);
70     }
71     else if (constantType == typeof(long))
72     {
73         emitter.LoadConstant((long)constantValue);
74     }
75     else if (constantType == typeof(ulong))
76     {
77         emitter.LoadConstant((ulong)constantValue);
78     }
79     else if (constantType == typeof(int))
80     {
81         emitter.LoadConstant((int)constantValue);
82     }
83     else if (constantType == typeof(uint))
84     {
85         emitter.LoadConstant((uint)constantValue);
86     }
87     else if (constantType == typeof(short))
88     {
89         emitter.LoadConstant((short)constantValue);
90         emitter.Convert<short>();
91     }
92     else if (constantType == typeof(ushort))
93     {
94         emitter.LoadConstant((ushort)constantValue);
95         emitter.Convert<ushort>();
96     }
97     else if (constantType == typeof(sbyte))
98     {
99         emitter.LoadConstant((sbyte)constantValue);
100        emitter.Convert<sbyte>();
101    }
102    else if (constantType == typeof(byte))
103    {
104        emitter.LoadConstant((byte)constantValue);
105        emitter.Convert<byte>();
106    }
107    else
108    {
109        throw new NotSupportedException();
110    }
111    return emitter;
112 }
113
114 public static Emit<TDelegate> Increment<TDelegate>(this Emit<TDelegate> emitter, Type
→ valueType)
115 {
116     emitter.LoadConstantOne(valueType);

```

```

117         emitter.Add();
118         return emitter;
119     }
120
121     public static Emit<TDelegate> Decrement<TDelegate>(this Emit<TDelegate> emitter, Type
    ↪ valueType)
122     {
123         emitter.LoadConstantOne(valueType);
124         emitter.Subtract();
125         return emitter;
126     }
127
128     public static Emit<TDelegate> LoadArguments<TDelegate>(this Emit<TDelegate> emitter,
    ↪ params ushort[] arguments)
129     {
130         for (var i = 0; i < arguments.Length; i++)
131         {
132             emitter.LoadArgument(arguments[i]);
133         }
134         return emitter;
135     }
136
137     public static Emit<TDelegate> CompareGreaterThan<TDelegate>(this Emit<TDelegate> emitter,
    ↪ bool isSigned)
138     {
139         if (isSigned)
140         {
141             emitter.CompareGreaterThan();
142         }
143         else
144         {
145             emitter.UnsignedCompareGreaterThan();
146         }
147         return emitter;
148     }
149
150     public static Emit<TDelegate> CompareLessThan<TDelegate>(this Emit<TDelegate> emitter,
    ↪ bool isSigned)
151     {
152         if (isSigned)
153         {
154             emitter.CompareLessThan();
155         }
156         else
157         {
158             emitter.UnsignedCompareLessThan();
159         }
160         return emitter;
161     }
162
163     public static Emit<TDelegate> BranchIfGreaterOrEqual<TDelegate>(this Emit<TDelegate>
    ↪ emitter, bool isSigned, Label label)
164     {
165         if (isSigned)
166         {
167             emitter.BranchIfGreaterOrEqual(label);
168         }
169         else
170         {
171             emitter.UnsignedBranchIfGreaterOrEqual(label);
172         }
173         return emitter;
174     }
175
176     public static Emit<TDelegate> BranchIfLessOrEqual<TDelegate>(this Emit<TDelegate>
    ↪ emitter, bool isSigned, Label label)
177     {
178         if (isSigned)
179         {
180             emitter.BranchIfLessOrEqual(label);
181         }
182         else
183         {
184             emitter.UnsignedBranchIfLessOrEqual(label);
185         }
186         return emitter;
187     }
188 }

```

189 }

./NotSupportedExceptionDelegateFactory.cs

```
1  using System;
2  using Sigil;
3  using Platform.Interfaces;
4
5  namespace Platform.Reflection.Sigil
6  {
7      public class NotSupportedExceptionDelegateFactory<TDelegate> : IFactory<TDelegate>
8      {
9          public TDelegate Create()
10         {
11             var emitter = Emit<TDelegate>.NewDynamicMethod();
12             emitter.NewObject<NotSupportedException>();
13             emitter.Throw();
14             return emitter.CreateDelegate();
15         }
16     }
17 }
```

Index

./DelegateHelpers.cs, 1
./EmitExtensions.cs, 1
./NotSupportedExceptionDelegateFactory.cs, 4