./DelegateHelpers.cs

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

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;
        }
    }
}
```

./EmitExtensions.cs

```
using System;
using Sigil;

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;
        }

        public static Emit<TDelegate> Increment<TDelegate>(this Emit<TDelegate> emiter, Type
        ↪  valueType)
        {
            emiter.LoadConstantOne(valueType);
```

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

```
}
```

./NotSupportedExceptionDelegateFactory.cs

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

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();
        }
    }
}
```

Index