

LinksPlatform's Platform.Data.Sequences Class Library

1.1 ./csharp/Platform.Data.Sequences/ISequenceAppender.cs

```
1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Sequences
6 {
7     public interface ISequenceAppender<TLinkAddress>
8     {
9         [MethodImpl(MethodImplOptions.AggressiveInlining)]
10         TLinkAddress Append(TLinkAddress sequence, TLinkAddress appendant);
11     }
12 }
```

1.2 ./csharp/Platform.Data.Sequences/ISequenceWalker.cs

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Sequences
7 {
8     public interface ISequenceWalker<TLinkAddress>
9     {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         IEnumerable<IList<TLinkAddress>> Walk(TLinkAddress sequence);
12     }
13 }
```

1.3 ./csharp/Platform.Data.Sequences/SequenceWalker.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Sequences
8 {
9     /// <remarks>
10     /// Реализованный внутри алгоритм наглядно показывает,
11     /// что совершенно не обязательна рекурсивная реализация (с вложенным вызовом функцией самой
12     /// ↪ себя),
13     /// так как стек можно использовать намного эффективнее при ручном управлении.
14     ///
15     /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
16     /// Решить встраивать ли защиту от заикливания.
17     /// Альтернативой защиты от закливания может быть заранее известное ограничение на
18     /// ↪ погружение вглубь.
19     /// А так же качественное распознавание прохода по циклическому графу.
20     /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
21     /// ↪ стека.
22     /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
23     /// </remarks>
24     public static class SequenceWalker
25     {
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public static void WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
28         ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
29         ↪ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
30         {
31             var stack = new Stack<TLinkAddress>();
32             var element = sequence;
33             if (isElement(element))
34             {
35                 visit(element);
36             }
37             else
38             {
39                 while (true)
40                 {
41                     if (isElement(element))
42                     {
43                         if (stack.Count == 0)
44                         {
45                             break;
46                         }
47                         element = stack.Pop();
48                         var source = getSource(element);
49                     }
50                 }
51             }
52         }
53     }
54 }
```

```

44         var target = getTarget(element);
45         if (isElement(source))
46         {
47             visit(source);
48         }
49         if (isElement(target))
50         {
51             visit(target);
52         }
53         element = target;
54     }
55     else
56     {
57         stack.Push(element);
58         element = getSource(element);
59     }
60 }
61 }
62 }
63
64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 public static void WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↪ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
66 {
67     var stack = new Stack<TLinkAddress>();
68     var element = sequence;
69     if (isElement(element))
70     {
71         visit(element);
72     }
73     else
74     {
75         while (true)
76         {
77             if (isElement(element))
78             {
79                 if (stack.Count == 0)
80                 {
81                     break;
82                 }
83                 element = stack.Pop();
84                 var source = getSource(element);
85                 var target = getTarget(element);
86                 if (isElement(target))
87                 {
88                     visit(target);
89                 }
90                 if (isElement(source))
91                 {
92                     visit(source);
93                 }
94                 element = source;
95             }
96             else
97             {
98                 stack.Push(element);
99                 element = getTarget(element);
100             }
101         }
102     }
103 }
104 }
105 }

```

1.4 ./csharp/Platform.Data.Sequences/StopableSequenceWalker.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Sequences
8 {
9     /// <remarks>
10     /// Реализованный внутри алгоритм наглядно показывает,
11     /// что совершенно не обязательна рекурсивная реализация (с вложенным вызовом функцией самой
    ↪ себя),
12     /// так как стек можно использовать намного эффективнее при ручном управлении.

```

```

13  ///
14  /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализации?
15  /// Решить встраивать ли защиту от заикливания.
16  /// Альтернативой защиты от заикливания может быть заранее известное ограничение на
    ↳ погружение вглубь.
17  /// А так же качественное распознавание прохода по циклическому графу.
18  /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
    ↳ стека.
19  /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
20  /// </remarks>
21  public static class StopableSequenceWalker
22  {
23      [MethodImpl(MethodImplOptions.AggressiveInlining)]
24      public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↳ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↳ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> enter, Action<TLinkAddress>
    ↳ exit, Func<TLinkAddress, bool> canEnter, Func<TLinkAddress, bool> visit)
25  {
26      var exited = 0;
27      var stack = new Stack<TLinkAddress>();
28      var element = sequence;
29      if (isElement(element))
30      {
31          return visit(element);
32      }
33      while (true)
34      {
35          if (isElement(element))
36          {
37              if (stack.Count == 0)
38              {
39                  return true;
40              }
41              element = stack.Pop();
42              exit(element);
43              exited++;
44              var source = getSource(element);
45              var target = getTarget(element);
46              if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
    ↳ !visit(source))
47              {
48                  return false;
49              }
50              if ((isElement(target) || !canEnter(target)) && !visit(target))
51              {
52                  return false;
53              }
54              element = target;
55          }
56          else
57          {
58              if (canEnter(element))
59              {
60                  enter(element);
61                  exited = 0;
62                  stack.Push(element);
63                  element = getSource(element);
64              }
65              else
66              {
67                  if (stack.Count == 0)
68                  {
69                      return true;
70                  }
71                  element = stack.Pop();
72                  exit(element);
73                  exited++;
74                  var source = getSource(element);
75                  var target = getTarget(element);
76                  if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
    ↳ !visit(source))
77                  {
78                      return false;
79                  }
80                  if ((isElement(target) || !canEnter(target)) && !visit(target))
81                  {
82                      return false;
83                  }
84                  element = target;

```

```

85     }
86 }
87 }
88 }
89
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↳ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↳ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
92 {
93     var stack = new Stack<TLinkAddress>();
94     var element = sequence;
95     if (isElement(element))
96     {
97         return visit(element);
98     }
99     while (true)
100     {
101         if (isElement(element))
102         {
103             if (stack.Count == 0)
104             {
105                 return true;
106             }
107             element = stack.Pop();
108             var source = getSource(element);
109             var target = getTarget(element);
110             if (isElement(source) && !visit(source))
111             {
112                 return false;
113             }
114             if (isElement(target) && !visit(target))
115             {
116                 return false;
117             }
118             element = target;
119         }
120         else
121         {
122             stack.Push(element);
123             element = getSource(element);
124         }
125     }
126 }
127
128 [MethodImpl(MethodImplOptions.AggressiveInlining)]
129 public static bool WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↳ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↳ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
130 {
131     var stack = new Stack<TLinkAddress>();
132     var element = sequence;
133     if (isElement(element))
134     {
135         return visit(element);
136     }
137     while (true)
138     {
139         if (isElement(element))
140         {
141             if (stack.Count == 0)
142             {
143                 return true;
144             }
145             element = stack.Pop();
146             var source = getSource(element);
147             var target = getTarget(element);
148             if (isElement(target) && !visit(target))
149             {
150                 return false;
151             }
152             if (isElement(source) && !visit(source))
153             {
154                 return false;
155             }
156             element = source;
157         }
158         else
159         {

```

```
160         stack.Push(element);
161         element = getTarget(element);
162     }
163 }
164 }
165 }
166 }
```

Index

- ./csharp/Platform.Data.Sequences/ISequenceAppender.cs, 1
- ./csharp/Platform.Data.Sequences/ISequenceWalker.cs, 1
- ./csharp/Platform.Data.Sequences/SequenceWalker.cs, 1
- ./csharp/Platform.Data.Sequences/StopableSequenceWalker.cs, 2