

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 стэк можно использовать намного эффективнее при ручном управлении.
14    ///
15    /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализаций?
16    /// Решить встраивать ли защиту от зацикливания.
17    /// Альтернативой защиты от заклинивания 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                        if (getTarget(source) != null)
50                        {
51                            stack.Push(source);
52                            visit(source);
53                        }
54                    }
55                }
56            }
57        }
58    }
59}
```

```

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,
66     → TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
67     → Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
68 {
69     var stack = new Stack<TLinkAddress>();
70     var element = sequence;
71     if (isElement(element))
72     {
73         visit(element);
74     }
75     else
76     {
77         while (true)
78         {
79             if (isElement(element))
80             {
81                 if (stack.Count == 0)
82                 {
83                     break;
84                 }
85                 element = stack.Pop();
86                 var source = getSource(element);
87                 var target = getTarget(element);
88                 if (isElement(target))
89                 {
90                     visit(target);
91                 }
92                 if (isElement(source))
93                 {
94                     visit(source);
95                 }
96                 element = source;
97             }
98             else
99             {
100                 stack.Push(element);
101                 element = getTarget(element);
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
12    → себя),
13    /// так как стэк можно использовать намного эффективнее при ручном управлении.

```

```

13  ///
14  /// Решить объединять ли логику в одну функцию, или оставить 4 отдельных реализаций?
15  /// Решить встраивать ли защиту от зацикливания.
16  /// Альтернативой защиты от зацикливания может быть заранее известное ограничение на
17  →    погружение вглубь.
18  /// А так же качественное распознавание прохода по циклическому графу.
19  /// Ограничение на уровень глубины рекурсии может позволить использовать уменьшенный размер
20  →    стека.
21  /// Можно использовать глобальный стек (или несколько глобальных стеков на каждый поток).
22  /// </remarks>
23  public static class StopableSequenceWalker
{
24      [MethodImpl(MethodImplOptions.AggressiveInlining)]
25      public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
26          → TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
27          → Func<TLinkAddress, bool> isElement, Action<TLinkAddress> enter, Action<TLinkAddress>
28          → exit, Func<TLinkAddress, bool> canEnter, Func<TLinkAddress, bool> visit)
29      {
30          var exited = 0;
31          var stack = new Stack<TLinkAddress>();
32          var element = sequence;
33          if (isElement(element))
34          {
35              return visit(element);
36          }
37          while (true)
38          {
39              if (isElement(element))
40              {
41                  if (stack.Count == 0)
42                  {
43                      return true;
44                  }
45                  element = stack.Pop();
46                  exit(element);
47                  exited++;
48                  var source = getSource(element);
49                  var target = getTarget(element);
50                  if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
51                      → !visit(source))
52                  {
53                      return false;
54                  }
55                  if ((isElement(target) || !canEnter(target)) && !visit(target))
56                  {
57                      return false;
58                  }
59                  element = target;
60              }
61              else
62              {
63                  if (canEnter(element))
64                  {
65                      enter(element);
66                      exited = 0;
67                      stack.Push(element);
68                      element = getSource(element);
69                  }
70              }
71          }
72          if (stack.Count == 0)
73          {
74              return true;
75          }
76          element = stack.Pop();
77          exit(element);
78          exited++;
79          var source = getSource(element);
80          var target = getTarget(element);
81          if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
82              → !visit(source))
83          {
84              return false;
85          }
86          if ((isElement(target) || !canEnter(target)) && !visit(target))
87          {
88              return false;
89          }
90          element = target;

```

```

85
86
87
88
89
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
92   → TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
93   → Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
94 {
95     var stack = new Stack<TLinkAddress>();
96     var element = sequence;
97     if (isElement(element))
98     {
99         return visit(element);
100    }
101   while (true)
102   {
103     if (isElement(element))
104     {
105         if (stack.Count == 0)
106         {
107             return true;
108         }
109         element = stack.Pop();
110         var source = getSource(element);
111         var target = getTarget(element);
112         if (isElement(source) && !visit(source))
113         {
114             return false;
115         }
116         if (isElement(target) && !visit(target))
117         {
118             return false;
119         }
120         element = target;
121     }
122     else
123     {
124         stack.Push(element);
125         element = getSource(element);
126     }
127   }
128
129 [MethodImpl(MethodImplOptions.AggressiveInlining)]
130 public static bool WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
131   → TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
132   → Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
133 {
134     var stack = new Stack<TLinkAddress>();
135     var element = sequence;
136     if (isElement(element))
137     {
138         return visit(element);
139     }
140     while (true)
141     {
142       if (isElement(element))
143       {
144           if (stack.Count == 0)
145           {
146               return true;
147           }
148           element = stack.Pop();
149           var source = getSource(element);
150           var target = getTarget(element);
151           if (isElement(target) && !visit(target))
152           {
153               return false;
154           }
155           if (isElement(source) && !visit(source))
156           {
157               return false;
158           }
159           element = source;
160       }
161     }

```

```
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