

## 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     /// <summary>
8     /// <para>
9     /// Defines the sequence appender.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    public interface ISequenceAppender<TLinkAddress>
14    {
15        /// <summary>
16        /// <para>
17        /// Appends the sequence.
18        /// </para>
19        /// <para></para>
20        /// </summary>
21        /// <param name="sequence">
22        /// <para>The sequence.</para>
23        /// <para></para>
24        /// </param>
25        /// <param name="appendant">
26        /// <para>The appendant.</para>
27        /// <para></para>
28        /// </param>
29        /// <returns>
30        /// <para>The link address</para>
31        /// <para></para>
32        /// </returns>
33        [MethodImpl(MethodImplOptions.AggressiveInlining)]
34        TLinkAddress Append(TLinkAddress sequence, TLinkAddress appendant);
35    }
36 }
```

### 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     /// <summary>
9     /// <para>
10    /// Defines the sequence walker.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    public interface ISequenceWalker<TLinkAddress>
15    {
16        /// <summary>
17        /// <para>
18        /// Walks the sequence.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <param name="sequence">
23        /// <para>The sequence.</para>
24        /// <para></para>
25        /// </param>
26        /// <returns>
27        /// <para>An enumerable of i list t link address</para>
28        /// <para></para>
29        /// </returns>
30        [MethodImpl(MethodImplOptions.AggressiveInlining)]
31        IEnumerable<IList<TLinkAddress>> Walk(TLinkAddress sequence);
32    }
33 }
```

### 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        /// <summary>
27        /// <para>
28        /// Walks the right using the specified sequence.
29        /// </para>
30        /// </summary>
31        /// <typeparam name="TLinkAddress">
32        /// <para>The link address.</para>
33        /// </typeparam>
34        /// <param name="sequence">
35        /// <para>The sequence.</para>
36        /// </param>
37        /// <param name="getSource">
38        /// <para>The get source.</para>
39        /// </param>
40        /// <param name="getTarget">
41        /// <para>The get target.</para>
42        /// </param>
43        /// <param name="isElement">
44        /// <para>The is element.</para>
45        /// </param>
46        /// <param name="visit">
47        /// <para>The visit.</para>
48        /// </param>
49        [MethodImpl(MethodImplOptions.AggressiveInlining)]
50        public static void WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
51        ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
52        ↪ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
53        {
54            var stack = new Stack<TLinkAddress>();
55            var element = sequence;
56            if (isElement(element))
57            {
58                visit(element);
59            }
60            else
61            {
62                while (true)
63                {
64                    if (isElement(element))
65                    {
66                        if (stack.Count == 0)
67                        {
68                            break;
69                        }
70                        element = stack.Pop();
71                        var source = getSource(element);
72                        var target = getTarget(element);
73                        if (isElement(source))
74                        {
75

```

```

77         visit(source);
78     }
79     if (isElement(target))
80     {
81         visit(target);
82     }
83     element = target;
84 }
85 else
86 {
87     stack.Push(element);
88     element = getSource(element);
89 }
90 }
91 }
92 }
93
94 /// <summary>
95 /// <para>
96 /// Walks the left using the specified sequence.
97 /// </para>
98 /// <para></para>
99 /// </summary>
100 /// <typeparam name="TLinkAddress">
101 /// <para>The link address.</para>
102 /// <para></para>
103 /// </typeparam>
104 /// <param name="sequence">
105 /// <para>The sequence.</para>
106 /// <para></para>
107 /// </param>
108 /// <param name="getSource">
109 /// <para>The get source.</para>
110 /// <para></para>
111 /// </param>
112 /// <param name="getTarget">
113 /// <para>The get target.</para>
114 /// <para></para>
115 /// </param>
116 /// <param name="isElement">
117 /// <para>The is element.</para>
118 /// <para></para>
119 /// </param>
120 /// <param name="visit">
121 /// <para>The visit.</para>
122 /// <para></para>
123 /// </param>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 public static void WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
    ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
    ↪ Func<TLinkAddress, bool> isElement, Action<TLinkAddress> visit)
126 {
127     var stack = new Stack<TLinkAddress>();
128     var element = sequence;
129     if (isElement(element))
130     {
131         visit(element);
132     }
133     else
134     {
135         while (true)
136         {
137             if (isElement(element))
138             {
139                 if (stack.Count == 0)
140                 {
141                     break;
142                 }
143                 element = stack.Pop();
144                 var source = getSource(element);
145                 var target = getTarget(element);
146                 if (isElement(target))
147                 {
148                     visit(target);
149                 }
150                 if (isElement(source))
151                 {
152                     visit(source);

```

```

153     }
154     element = source;
155 }
156 else
157 {
158     stack.Push(element);
159     element = getTarget(element);
160 }
161 }
162 }
163 }
164 }
165 }

```

#### 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 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 StopableSequenceWalker
25    {
26        /// <summary>
27        /// <para>
28        /// Determines whether walk right.
29        /// </para>
30        /// </summary>
31        /// <typeparam name="TLinkAddress">
32        /// <para>The link address.</para>
33        /// </typeparam>
34        /// <param name="sequence">
35        /// <para>The sequence.</para>
36        /// </param>
37        /// <param name="getSource">
38        /// <para>The get source.</para>
39        /// </param>
40        /// <param name="getTarget">
41        /// <para>The get target.</para>
42        /// </param>
43        /// <param name="isElement">
44        /// <para>The is element.</para>
45        /// </param>
46        /// <param name="enter">
47        /// <para>The enter.</para>
48        /// </param>
49        /// <param name="exit">
50        /// <para>The exit.</para>
51        /// </param>
52        /// <param name="canEnter">
53        /// <para>The can enter.</para>
54        /// </param>
55    }
56 }

```

```

61     /// <param name="visit">
62     /// <para>The visit.</para>
63     /// <para></para>
64     /// </param>
65     /// <returns>
66     /// <para>The bool</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     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)
71     {
72         var exited = 0;
73         var stack = new Stack<TLinkAddress>();
74         var element = sequence;
75         if (isElement(element))
76         {
77             return visit(element);
78         }
79         while (true)
80         {
81             if (isElement(element))
82             {
83                 if (stack.Count == 0)
84                 {
85                     return true;
86                 }
87                 element = stack.Pop();
88                 exit(element);
89                 exited++;
90                 var source = getSource(element);
91                 var target = getTarget(element);
92                 if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
    ↪ !visit(source))
93                 {
94                     return false;
95                 }
96                 if ((isElement(target) || !canEnter(target)) && !visit(target))
97                 {
98                     return false;
99                 }
100                 element = target;
101             }
102             else
103             {
104                 if (canEnter(element))
105                 {
106                     enter(element);
107                     exited = 0;
108                     stack.Push(element);
109                     element = getSource(element);
110                 }
111                 else
112                 {
113                     if (stack.Count == 0)
114                     {
115                         return true;
116                     }
117                     element = stack.Pop();
118                     exit(element);
119                     exited++;
120                     var source = getSource(element);
121                     var target = getTarget(element);
122                     if ((isElement(source) || (exited == 1 && !canEnter(source))) &&
    ↪ !visit(source))
123                     {
124                         return false;
125                     }
126                     if ((isElement(target) || !canEnter(target)) && !visit(target))
127                     {
128                         return false;
129                     }
130                     element = target;
131                 }
132             }
133         }
134     }

```

```

135     /// <summary>
136     /// <para>
137     /// Determines whether walk right.
138     /// </para>
139     /// </summary>
140     /// <param name="TLinkAddress">
141     /// <para>The link address.</para>
142     /// </param>
143     /// <param name="sequence">
144     /// <para>The sequence.</para>
145     /// </param>
146     /// <param name="getSource">
147     /// <para>The get source.</para>
148     /// </param>
149     /// <param name="getTarget">
150     /// <para>The get target.</para>
151     /// </param>
152     /// <param name="isElement">
153     /// <para>The is element.</para>
154     /// </param>
155     /// <param name="visit">
156     /// <para>The visit.</para>
157     /// </param>
158     /// <returns>
159     /// <para>The bool</para>
160     /// </returns>
161     [MethodImpl(MethodImplOptions.AggressiveInlining)]
162     public static bool WalkRight<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
163     ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
164     ↪ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
165     {
166         var stack = new Stack<TLinkAddress>();
167         var element = sequence;
168         if (isElement(element))
169         {
170             return visit(element);
171         }
172         while (true)
173         {
174             if (isElement(element))
175             {
176                 if (stack.Count == 0)
177                 {
178                     return true;
179                 }
180                 element = stack.Pop();
181                 var source = getSource(element);
182                 var target = getTarget(element);
183                 if (isElement(source) && !visit(source))
184                 {
185                     return false;
186                 }
187                 if (isElement(target) && !visit(target))
188                 {
189                     return false;
190                 }
191                 element = target;
192             }
193             else
194             {
195                 stack.Push(element);
196                 element = getSource(element);
197             }
198         }
199     }
200
201     /// <summary>
202     /// <para>
203     /// Determines whether walk left.
204

```

```

211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <typeparam name="TLinkAddress">
215     /// <para>The link address.</para>
216     /// <para></para>
217     /// </typeparam>
218     /// <param name="sequence">
219     /// <para>The sequence.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="getSource">
223     /// <para>The get source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="getTarget">
227     /// <para>The get target.</para>
228     /// <para></para>
229     /// </param>
230     /// <param name="isElement">
231     /// <para>The is element.</para>
232     /// <para></para>
233     /// </param>
234     /// <param name="visit">
235     /// <para>The visit.</para>
236     /// <para></para>
237     /// </param>
238     /// <returns>
239     /// <para>The bool</para>
240     /// <para></para>
241     /// </returns>
242     [MethodImpl(MethodImplOptions.AggressiveInlining)]
243     public static bool WalkLeft<TLinkAddress>(TLinkAddress sequence, Func<TLinkAddress,
        ↪ TLinkAddress> getSource, Func<TLinkAddress, TLinkAddress> getTarget,
        ↪ Func<TLinkAddress, bool> isElement, Func<TLinkAddress, bool> visit)
244     {
245         var stack = new Stack<TLinkAddress>();
246         var element = sequence;
247         if (isElement(element))
248         {
249             return visit(element);
250         }
251         while (true)
252         {
253             if (isElement(element))
254             {
255                 if (stack.Count == 0)
256                 {
257                     return true;
258                 }
259                 element = stack.Pop();
260                 var source = getSource(element);
261                 var target = getTarget(element);
262                 if (isElement(target) && !visit(target))
263                 {
264                     return false;
265                 }
266                 if (isElement(source) && !visit(source))
267                 {
268                     return false;
269                 }
270                 element = source;
271             }
272             else
273             {
274                 stack.Push(element);
275                 element = getTarget(element);
276             }
277         }
278     }
279 }
280 }

```

## 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, 4