1.1 ./csharp/Platform.Threading/ConcurrentQueueExtensions.cs

```
using System;
using System.Collections.Concurrent;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using Platform.Collections.Concurrent;

namespace Platform.Threading
{
    /// <summary>
    /// <para>Provides a set of extension methods for <see cref="ConcurrentQueue{T}"/>
    ↪  objects.</para>
    /// <para>Предоставляет набор методов расширения для объектов <see
    ↪  cref="ConcurrentQueue{T}"/>.</para>
    /// </summary>
    public static class ConcurrentQueueExtensions
    {
        /// <summary>
        /// <para>Suspends evaluation of the method until all asynchronous operations in the
        ↪  queue finish.</para>
        /// <para>Приостановляет выполнение метода до завершения всех асинхронных операций в
        ↪  очереди.</para>
        /// </summary>
        /// <param name="queue"><para>The queue of asynchronous operations.</para><para>Очередь
        ↪  асинхронных операций.</para></param>
        /// <returns><para>An asynchronous operation representation.</para><para>Представление
        ↪  асинхронной операции.</para></returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static async Task AwaitAll(this ConcurrentQueue<Task> queue)
        {
            foreach (var item in queue.DequeueAll())
            {
                await item.ConfigureAwait(continueOnCapturedContext: false);
            }
        }

        /// <summary>
        /// <para>Suspends evaluation of the method until the first asynchronous operation in
        ↪  the queue finishes.</para>
        /// <para>Приостанавливает выполнение метода до завершения первой асинхронной операции в
        ↪  очереди.</para>
        /// </summary>
        /// <param name="queue"><para>The queue of asynchronous operations.</para><para>Очередь
        ↪  асинхронных операций.</para></param>
        /// <returns><para>An asynchronous operation representation.</para><para>Представление
        ↪  асинхронной операции.</para></returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static async Task AwaitOne(this ConcurrentQueue<Task> queue)
        {
            if (queue.TryDequeue(out Task item))
            {
                await item.ConfigureAwait(continueOnCapturedContext: false);
            }
        }

        /// <summary>
        /// <para>Adds an <see cref="Action"/> as runned <see cref="Task"/> to the end of the
        ↪  <see cref="ConcurrentQueue{T}"/>.</para>
        /// <para>Добавляет <see cref="Action"/> как запущенную <see cref="Task"/> в конец <see
        ↪  cref="ConcurrentQueue{T}"/>.</para>
        /// </summary>
        /// <param name="queue"><para>The queue of asynchronous operations.</para><para>Очередь
        ↪  асинхронных операций.</para></param>
        /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делагат
        ↪  <see cref="Action"/>.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void RunAndPush(this ConcurrentQueue<Task> queue, Action action) =>
        ↪  queue.Enqueue(Task.Run(action));
    }
}
```

1.2 ./csharp/Platform.Threading/Synchronization/ISynchronization.cs

```
using System;
using System.Runtime.CompilerServices;

namespace Platform.Threading.Synchronization
```

```
{
    /// <summary>
    /// <para>Represents a synchronization object that supports read and write operations.</para>
    /// <para>Представляет объект синхронизации с поддержкой операций чтения и записи.</para>
    /// </summary>
    public interface ISynchronization
    {
        /// <summary>
        /// <para>Executes action in read access mode.</para>
        /// <para>Выполняет действие в режиме доступа для чтения.</para>
        /// </summary>
        /// <param name="action"><para>The action.</para><para>Действие.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        void DoRead(Action action);

        /// <summary>
        /// <para>Executes a function in read access mode and returns the function's result.</para>
        /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё результат.</para>
        /// </summary>
        /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип результата функции.</para></typeparam>
        /// <param name="function"><para>The function.</para><para>Функция.</para></param>
        /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        TResult DoRead<TResult>(Func<TResult> function);

        /// <summary>
        /// <para>Executes action in write access mode.</para>
        /// <para>Выполняет действие в режиме доступа для записи.</para>
        /// </summary>
        /// <param name="action"><para>The action.</para><para>Действие.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        void DoWrite(Action action);

        /// <summary>
        /// <para>Executes a function in write access mode and returns the function's result.</para>
        /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё результат.</para>
        /// </summary>
        /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип результата функции.</para></typeparam>
        /// <param name="function"><para>The function.</para><para>Функция.</para></param>
        /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        TResult DoWrite<TResult>(Func<TResult> function);
    }
}
```

1.3 ./csharp/Platform.Threading/Synchronization/ISynchronizationExtensions.cs

```
using System;
using System.Runtime.CompilerServices;

namespace Platform.Threading.Synchronization
{
    /// <summary>
    /// <para>Contains extension methods for the <see cref="ISynchronization"/> interface.</para>
    /// <para>Содержит методы расширения для интерфейса <see cref="ISynchronization"/>.</para>
    /// </summary>
    public static class ISynchronizationExtensions
    {
        /// <summary>
        /// <para>Executes a function in read access mode and returns the function's result.</para>
        /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё результат.</para>
        /// </summary>
        /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип результата функции.</para></typeparam>
        /// <typeparam name="TParam"><para>The parameter type.</para><para>Тип параметра.</para></typeparam>
        /// <param name="synchronization"><para>Synchronization object.</para><para>Синхронизация объекта.</para></param>
        /// <param name="parameter"><para>The parameter</para><para>Параметр.</para></param>
        /// <param name="function"><para>The function.</para><para>Функция.</para></param>
```

```
21          /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
22          [MethodImpl(MethodImplOptions.AggressiveInlining)]
23          public static TResult DoRead<TResult, TParam>(this ISynchronization synchronization,
            ↪  TParam parameter, Func<TParam, TResult> function) => synchronization.DoRead(() =>
            ↪  function(parameter));
24
25          /// <summary>
26          /// <para>Executes action in read access mode.</para>
27          /// <para>Выполняет действие в режиме доступа для чтения.</para>
28          /// </summary>
29          /// <typeparam name="TParam"><para>The parameter type.</para><para>Тип
            ↪  параметра.</para></typeparam>
30          /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
31          /// <param name="parameter"><para>The parameter</para><para>Параметр.</para></param>
32          /// <param name="action"><para>The action.</para><para>Действие.</para></param>
33          [MethodImpl(MethodImplOptions.AggressiveInlining)]
34          public static void DoRead<TParam>(this ISynchronization synchronization, TParam
            ↪  parameter, Action<TParam> action) => synchronization.DoRead(() => action(parameter));
35
36          /// <summary>
37          /// <para>Executes a function in write access mode and returns the function's
            ↪  result.</para>
38          /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
            ↪  результат.</para>
39          /// </summary>
40          /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
41          /// <typeparam name="TParam"><para>The parameter type.</para><para>Тип
            ↪  параметра.</para></typeparam>
42          /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
43          /// <param name="parameter"><para>The parameter</para><para>Параметр.</para></param>
44          /// <param name="function"><para>The function.</para><para>Функция.</para></param>
45          /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
46          [MethodImpl(MethodImplOptions.AggressiveInlining)]
47          public static TResult DoWrite<TResult, TParam>(this ISynchronization synchronization,
            ↪  TParam parameter, Func<TParam, TResult> function) => synchronization.DoWrite(() =>
            ↪  function(parameter));
48
49          /// <summary>
50          /// <para>Executes action in write access mode.</para>
51          /// <para>Выполняет действие в режиме доступа для записи.</para>
52          /// </summary>
53          /// <typeparam name="TParam"><para>The parameter type.</para><para>Тип
            ↪  параметра.</para></typeparam>
54          /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
55          /// <param name="parameter"><para>The parameter</para><para>Параметр.</para></param>
56          /// <param name="action"><para>The action.</para><para>Действие.</para></param>
57          [MethodImpl(MethodImplOptions.AggressiveInlining)]
58          public static void DoWrite<TParam>(this ISynchronization synchronization, TParam
            ↪  parameter, Action<TParam> action) => synchronization.DoWrite(() =>
            ↪  action(parameter));
59
60          /// <summary>
61          /// <para>Executes a function in read access mode and returns the function's
            ↪  result.</para>
62          /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё
            ↪  результат.</para>
63          /// </summary>
64          /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
65          /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
66          /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
67          /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
68          /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
69          /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
70          /// <param name="function"><para>The function.</para><para>Функция.</para></param>
71          /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
72          [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
73          public static TResult DoRead<TResult, TParam1, TParam2>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, Func<TParam1, TParam2,
            ↪  TResult> function) => synchronization.DoRead(() => function(parameter1, parameter2));
74
75          /// <summary>
76          /// <para>Executes action in read access mode.</para>
77          /// <para>Выполняет действие в режиме доступа для чтения.</para>
78          /// </summary>
79          /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
80          /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
81          /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
82          /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
83          /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
84          /// <param name="action"><para>The action.</para><para>Действие.</para></param>
85          [MethodImpl(MethodImplOptions.AggressiveInlining)]
86          public static void DoRead<TParam1, TParam2>(this ISynchronization synchronization,
            ↪  TParam1 parameter1, TParam2 parameter2, Action<TParam1, TParam2> action) =>
            ↪  synchronization.DoRead(() => action(parameter1, parameter2));
87
88          /// <summary>
89          /// <para>Executes a function in write access mode and returns the function's
            ↪  result.</para>
90          /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
            ↪  результат.</para>
91          /// </summary>
92          /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
93          /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
94          /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
95          /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
96          /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
97          /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
98          /// <param name="function"><para>The function.</para><para>Функция.</para></param>
99          /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
100         [MethodImpl(MethodImplOptions.AggressiveInlining)]
101         public static TResult DoWrite<TResult, TParam1, TParam2>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, Func<TParam1, TParam2,
            ↪  TResult> function) => synchronization.DoWrite(() => function(parameter1,
            ↪  parameter2));
102
103         /// <summary>
104         /// <para>Executes action in write access mode.</para>
105         /// <para>Выполняет действие в режиме доступа для записи.</para>
106         /// </summary>
107         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
108         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
109         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
110         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
111         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
112         /// <param name="action"><para>The action.</para><para>Действие.</para></param>
113         [MethodImpl(MethodImplOptions.AggressiveInlining)]
114         public static void DoWrite<TParam1, TParam2>(this ISynchronization synchronization,
            ↪  TParam1 parameter1, TParam2 parameter2, Action<TParam1, TParam2> action) =>
            ↪  synchronization.DoWrite(() => action(parameter1, parameter2));
115
116         /// <summary>
117         /// <para>Executes a function in read access mode and returns the function's
            ↪  result.</para>
118         /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё
            ↪  результат.</para>
```

```
119         /// </summary>
120         /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
121         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
122         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
123         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
124         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
125         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
126         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
127         /// <param name="parameter3"><para>The third parameter</para><para>Третий
            ↪  параметр.</para></param>
128         /// <param name="function"><para>The function.</para><para>Функция.</para></param>
129         /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
130         [MethodImpl(MethodImplOptions.AggressiveInlining)]
131         public static TResult DoRead<TResult, TParam1, TParam2, TParam3>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, TParam3 parameter3,
            ↪  Func<TParam1, TParam2, TParam3, TResult> function) => synchronization.DoRead(() =>
            ↪  function(parameter1, parameter2, parameter3));
132
133         /// <summary>
134         /// <para>Executes action in read access mode.</para>
135         /// <para>Выполняет действие в режиме доступа для чтения.</para>
136         /// </summary>
137         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
138         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
139         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
140         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
141         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
142         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
143         /// <param name="parameter3"><para>The third parameter</para><para>Третий
            ↪  параметр.</para></param>
144         /// <param name="action"><para>The action.</para><para>Действие.</para></param>
145         [MethodImpl(MethodImplOptions.AggressiveInlining)]
146         public static void DoRead<TParam1, TParam2, TParam3>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, TParam3 parameter3,
            ↪  Action<TParam1, TParam2, TParam3> action) => synchronization.DoRead(() =>
            ↪  action(parameter1, parameter2, parameter3));
147
148         /// <summary>
149         /// <para>Executes a function in write access mode and returns the function's
            ↪  result.</para>
150         /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
            ↪  результат.</para>
151         /// </summary>
152         /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
153         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
154         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
155         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
156         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
157         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
158         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
159         /// <param name="parameter3"><para>The third parameter</para><para>Третий
            ↪  параметр.</para></param>
160         /// <param name="function"><para>The function.</para><para>Функция.</para></param>
161         /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
162         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
163         public static TResult DoWrite<TResult, TParam1, TParam2, TParam3>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, TParam3 parameter3,
            ↪  Func<TParam1, TParam2, TParam3, TResult> function) => synchronization.DoWrite(() =>
            ↪  function(parameter1, parameter2, parameter3));
164
165         /// <summary>
166         /// <para>Executes action in write access mode.</para>
167         /// <para>Выполняет действие в режиме доступа для записи.</para>
168         /// </summary>
169         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
170         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
171         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
172         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
173         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
174         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
175         /// <param name="parameter3"><para>The third parameter</para><para>Третий
            ↪  параметр.</para></param>
176         /// <param name="action"><para>The action.</para><para>Действие.</para></param>
177         [MethodImpl(MethodImplOptions.AggressiveInlining)]
178         public static void DoWrite<TParam1, TParam2, TParam3>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, TParam3 parameter3,
            ↪  Action<TParam1, TParam2, TParam3> action) => synchronization.DoWrite(() =>
            ↪  action(parameter1, parameter2, parameter3));
179
180         /// <summary>
181         /// <para>Executes a function in read access mode and returns the function's
            ↪  result.</para>
182         /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё
            ↪  результат.</para>
183         /// </summary>
184         /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
185         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
186         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
187         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
188         /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
            ↪  параметра.</para></typeparam>
189         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
190         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
191         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
192         /// <param name="parameter3"><para>The third parameter</para><para>Третий
            ↪  параметр.</para></param>
193         /// <param name="parameter4"><para>The forth parameter</para><para>Чертвёртый
            ↪  параметр.</para></param>
194         /// <param name="function"><para>The function.</para><para>Функция.</para></param>
195         /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
196         [MethodImpl(MethodImplOptions.AggressiveInlining)]
197         public static TResult DoRead<TResult, TParam1, TParam2, TParam3, TParam4>(this
            ↪  ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2, TParam3
            ↪  parameter3, TParam4 parameter4, Func<TParam1, TParam2, TParam3, TParam4, TResult>
            ↪  function) => synchronization.DoRead(() => function(parameter1, parameter2,
            ↪  parameter3, parameter4));
198
199         /// <summary>
200         /// <para>Executes action in read access mode.</para>
201         /// <para>Выполняет действие в режиме доступа для чтения.</para>
202         /// </summary>
203         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
204         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
205         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
```

```
206         /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
            ↪  параметра.</para></typeparam>
207         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
208         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
209         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
210         /// <param name="parameter3"><para>The third parameter</para><para>Третий
            ↪  параметр.</para></param>
211         /// <param name="parameter4"><para>The forth parameter</para><para>Чертвёртый
            ↪  параметр.</para></param>
212         /// <param name="action"><para>The action.</para><para>Действие.</para></param>
213         [MethodImpl(MethodImplOptions.AggressiveInlining)]
214         public static void DoRead<TParam1, TParam2, TParam3, TParam4>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, TParam3 parameter3, TParam4
            ↪  parameter4, Action<TParam1, TParam2, TParam3, TParam4> action) =>
            ↪  synchronization.DoRead(() => action(parameter1, parameter2, parameter3, parameter4));
215
216         /// <summary>
217         /// <para>Executes a function in write access mode and returns the function's
            ↪  result.</para>
218         /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
            ↪  результат.</para>
219         /// </summary>
220         /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
221         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
222         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
223         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
224         /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
            ↪  параметра.</para></typeparam>
225         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
226         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
227         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
228         /// <param name="parameter3"><para>The third parameter</para><para>Третий
            ↪  параметр.</para></param>
229         /// <param name="parameter4"><para>The forth parameter</para><para>Чертвёртый
            ↪  параметр.</para></param>
230         /// <param name="function"><para>The function.</para><para>Функция.</para></param>
231         /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
232         [MethodImpl(MethodImplOptions.AggressiveInlining)]
233         public static TResult DoWrite<TResult, TParam1, TParam2, TParam3, TParam4>(this
            ↪  ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2, TParam3
            ↪  parameter3, TParam4 parameter4, Func<TParam1, TParam2, TParam3, TParam4, TResult>
            ↪  function) => synchronization.DoWrite(() => function(parameter1, parameter2,
            ↪  parameter3, parameter4));
234
235         /// <summary>
236         /// <para>Executes action in write access mode.</para>
237         /// <para>Выполняет действие в режиме доступа для записи.</para>
238         /// </summary>
239         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
240         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
241         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
242         /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
            ↪  параметра.</para></typeparam>
243         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
244         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
245         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
246         /// <param name="parameter3"><para>The third parameter</para><para>Третий
            ↪  параметр.</para></param>
```

```
247         /// <param name="parameter4"><para>The forth parameter</para><para>Чертвёртый
            ↪ параметр.</para></param>
248         /// <param name="action"><para>The action.</para><para>Действие.</para></param>
249         [MethodImpl(MethodImplOptions.AggressiveInlining)]
250         public static void DoWrite<TParam1, TParam2, TParam3, TParam4>(this ISynchronization
            ↪ synchronization, TParam1 parameter1, TParam2 parameter2, TParam3 parameter3, TParam4
            ↪ parameter4, Action<TParam1, TParam2, TParam3, TParam4> action) =>
            ↪ synchronization.DoWrite(() => action(parameter1, parameter2, parameter3,
            ↪ parameter4));
251     }
252 }
```

1.4 ./csharp/Platform.Threading/Synchronization/ISynchronized.cs

```
1  using System.Runtime.CompilerServices;
2
3  namespace Platform.Threading.Synchronization
4  {
5      /// <summary>
6      /// <para>Represents extendable synchronized interface access gate.</para>
7      /// <para>Представляет расширяемый интерфейс шлюза синхронизированного доступа.</para>
8      /// </summary>
9      /// <typeparam name="TInterface"><para>Synchronized interface.</para><para>Синхронизируемый
       ↪ интерфейс.</para></typeparam>
10     public interface ISynchronized<out TInterface>
11     {
12         /// <summary>
13         /// <para>Gets sychronization method.</para>
14         /// <para>Возвращает способ синхронизации.</para>
15         /// </summary>
16         ISynchronization SyncRoot
17         {
18             [MethodImpl(MethodImplOptions.AggressiveInlining)]
19             get;
20         }
21
22         /// <summary>
23         /// <para>Get source version of <typeparamref name="TInterface"/>, that does not
           ↪ garantee thread safe access synchronization.</para>
24         /// <para>Возвращает исходную версию <typeparamref name="TInterface"/>, которая не
           ↪ гарантирует потокобезопасную синхронизацию доступа.</para>
25         /// </summary>
26         /// <remarks>
27         /// <para>It is unsafe to use it directly, unless compound context using SyncRoot is
           ↪ created.</para>
28         /// <para>Использовать напрямую небезопасно, за исключением ситуации когда создаётся
           ↪ составной контекст с использованием SyncRoot.</para>
29         /// </remarks>
30         TInterface Unsync
31         {
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             get;
34         }
35
36         /// <summary>
37         /// <para>Get wrapped/decorated version of <typeparamref name="TInterface"/>, that does
           ↪ garantee thread safe access synchronization.</para>
38         /// <para>Возвращает обернутую/декорированную версию <typeparamref name="TInterface"/>,
           ↪ которая гарантирует потокобезопасную синхронизацию доступа.</para>
39         /// </summary>
40         /// <remarks>
41         /// <para>It is safe to use it directly, because it must be thread safe
           ↪ implementation.</para>
42         /// <para>Безопасно использовать напрямую, так как реализация должна быть
           ↪ потокобезопасной.</para>
43         /// </remarks>
44         TInterface Sync
45         {
46             [MethodImpl(MethodImplOptions.AggressiveInlining)]
47             get;
48         }
49     }
50 }
```

1.5 ./csharp/Platform.Threading/Synchronization/ReaderWriterLockSynchronization.cs

```
1  using System;
2  using System.Runtime.CompilerServices;
3  using System.Threading;
```

```
4
5   namespace Platform.Threading.Synchronization
6   {
7       /// <summary>
8       /// <para>Implementation of <see cref="ISynchronization"/> based on <see
        ↪  cref="ReaderWriterLockSlim"/>.</para>
9       /// <para>Реализация <see cref="ISynchronization"/> на основе <see
        ↪  cref="ReaderWriterLockSlim"/>.</para>
10      /// </summary>
11      public class ReaderWriterLockSynchronization : ISynchronization
12      {
13          private readonly ReaderWriterLockSlim _rwLock = new
            ↪  ReaderWriterLockSlim(LockRecursionPolicy.SupportsRecursion);
14
15          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪  path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪  n.DoRead(System.Action)"]/*'/>
16          /// <inheritdoc/>
17          [MethodImpl(MethodImplOptions.AggressiveInlining)]
18          public void DoRead(Action action)
19          {
20              _rwLock.EnterReadLock();
21              try
22              {
23                  action();
24              }
25              finally
26              {
27                  _rwLock.ExitReadLock();
28              }
29          }
30
31          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪  path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪  n.DoRead``1(System.Func{``0})"]/*'/>
32          /// <inheritdoc/>
33          [MethodImpl(MethodImplOptions.AggressiveInlining)]
34          public TResult DoRead<TResult>(Func<TResult> function)
35          {
36              _rwLock.EnterReadLock();
37              try
38              {
39                  return function();
40              }
41              finally
42              {
43                  _rwLock.ExitReadLock();
44              }
45          }
46
47          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪  path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪  n.DoWrite(System.Action)"]/*'/>
48          /// <inheritdoc/>
49          [MethodImpl(MethodImplOptions.AggressiveInlining)]
50          public void DoWrite(Action action)
51          {
52              _rwLock.EnterWriteLock();
53              try
54              {
55                  action();
56              }
57              finally
58              {
59                  _rwLock.ExitWriteLock();
60              }
61          }
62
63          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪  path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪  n.DoWrite``1(System.Func{``0})"]/*'/>
64          /// <inheritdoc/>
65          [MethodImpl(MethodImplOptions.AggressiveInlining)]
66          public TResult DoWrite<TResult>(Func<TResult> function)
67          {
68              _rwLock.EnterWriteLock();
69              try
70              {
```

```
                return function();
            }
            finally
            {
                _rwLock.ExitWriteLock();
            }
        }
    }
}
```

1.6 ./csharp/Platform.Threading/Synchronization/Unsynchronization.cs

```
using System;
using System.Runtime.CompilerServices;

namespace Platform.Threading.Synchronization
{
    /// <summary>
    /// <para>Implementation of <see cref="ISynchronization"/> that makes no actual synchronization.</para>
    /// <para>Реализация <see cref="ISynchronization"/>, которая не выполняет фактическую синхронизацию.</para>
    /// </summary>
    public class Unsynchronization : ISynchronization
    {
        /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml' path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronization.DoRead(System.Action)"]/*'/>
        /// <inheritdoc/>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void DoRead(Action action) => action();

        /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml' path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronization.DoRead``1(System.Func{``0})"]/*'/>
        /// <inheritdoc/>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TResult DoRead<TResult>(Func<TResult> function) => function();

        /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml' path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronization.DoWrite(System.Action)"]/*'/>
        /// <inheritdoc/>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void DoWrite(Action action) => action();

        /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml' path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronization.DoWrite``1(System.Func{``0})"]/*'/>
        /// <inheritdoc/>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TResult DoWrite<TResult>(Func<TResult> function) => function();
    }
}
```

1.7 ./csharp/Platform.Threading/TaskExtensions.cs

```
using System.Runtime.CompilerServices;
using System.Threading.Tasks;

namespace Platform.Threading
{
    /// <summary>
    /// <para>Provides a set of extension methods for <see cref="Task{TReturn}"/> objects.</para>
    /// <para>Предоставляет набор методов расширения для объектов <see cref="Task{TReturn}"/>.</para>
    /// </summary>
    public static class TaskExtensions
    {
        /// <summary>
        /// <para>Waits for completion of the asynchronous <see cref="Task{TReturn}"/> and returns its result.</para>
        /// <para>Ожидает завершения асинхронной <see cref="Task{TReturn}"/> и возвращает её результат.</para>
        /// </summary>
        /// <typeparam name="TReturn"><para>The return value type.</para><para>Тип возвращаемого значения.</para></typeparam>
        /// <param name="task"><para>The asynchronous <see cref="Task{TReturn}"/>.</para><para>Ассинхронная <see cref="Task{TReturn}"/>.</para></param>
```

```
        /// <returns><para>The result of completed <see cref="Task{TReturn}"/>.</para><para>Результат завершённой <see cref="Task{TReturn}"/>.</para></returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TReturn AwaitResult<TReturn>(this Task<TReturn> task) => task.GetAwaiter().GetResult();
    }
}
```

1.8 ./csharp/Platform.Threading/ThreadHelpers.cs

```
using System;
using System.Runtime.CompilerServices;
using System.Threading;

namespace Platform.Threading
{
    /// <summary>
    /// <para>Provides a set of helper methods for <see cref="Thread"/> objects.</para>
    /// <para>Предоставляет набор вспомогательных методов для объектов <see cref="Thread"/>.</para>
    /// </summary>
    public static class ThreadHelpers
    {
        /// <summary>
        /// <para>Gets the maximum stack size in bytes by default.</para>
        /// <para>Возвращает размер максимальный стека в байтах по умолчанию.</para>
        /// </summary>
        public static readonly int DefaultMaxStackSize;

        /// <summary>
        /// <para>Gets the extended maximum stack size in bytes by default.</para>
        /// <para>Возвращает расширенный максимальный размер стека в байтах по умолчанию.</para>
        /// </summary>
        public static readonly int DefaultExtendedMaxStackSize = 256 * 1024 * 1024;

        /// <summary>
        /// <para>Returns the default time interval for transferring control to other threads in milliseconds</para>
        /// <para>Возвращает интервал времени для передачи управления другим потокам в миллисекундах по умолчанию.</para>
        /// </summary>
        public static readonly int DefaultSleepInterval = 1;

        /// <summary>
        /// <para>Invokes the <see cref="Action{T}"/> with modified maximum stack size.</para>
        /// <para>Вызывает <see cref="Action{T}"/> с изменённым максимальным размером стека.</para>
        /// </summary>
        /// <typeparam name="T"><para>The type of the <see cref="Action{T}"/> argument.</para><para>Тип аргумента <see cref="Action{T}"/>.</para></typeparam>
        /// <param name="param"><para>The object containing data to be used by the invoked <see cref="Action{T}"/> delegate.</para><para>Объект, содержащий данные, которые будут использоваться вызваемым делегатом <see cref="Action{T}"/>.</para></param>
        /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делагат <see cref="Action{T}"/>.</para></param>
        /// <param name="maxStackSize"><para>The maximum stack size in bytes.</para><para>Максимальный размер стека в байтах.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void InvokeWithModifiedMaxStackSize<T>(T param, Action<object> action, int maxStackSize) => StartNew(param, action, maxStackSize).Join();

        /// <summary>
        /// <para>Invokes the <see cref="Action{T}"/> with extend maximum stack size.</para>
        /// <para>Вызывает <see cref="Action{T}"/> с расширенным максимальным размером стека.</para>
        /// </summary>
        /// <typeparam name="T"><para>The type of the <see cref="Action{T}"/> argument.</para><para>Тип аргумента <see cref="Action{T}"/>.</para></typeparam>
        /// <param name="param"><para>The object containing data to be used by the invoked <see cref="Action{T}"/> delegate.</para><para>Объект, содержащий данные, которые будут использоваться вызваемым делегатом <see cref="Action{T}"/>.</para></param>
        /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делагат <see cref="Action{T}"/>.</para></param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void InvokeWithExtendedMaxStackSize<T>(T param, Action<object> action) => InvokeWithModifiedMaxStackSize(param, action, DefaultExtendedMaxStackSize);

```

```
52          /// <summary>
53          /// <para>Invokes the <see cref="Action"/> with modified maximum stack size.</para>
54          /// <para>Вызывает <see cref="Action"/> с изменённым максимальным размером стека.</para>
55          /// </summary>
56          /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
            ↪  <see cref="Action"/>.</para></param>
57          /// <param name="maxStackSize"><para>The maximum stack size in
            ↪  bytes.</para><para>Максимальный размер стека в байтах.</para></param>
58          [MethodImpl(MethodImplOptions.AggressiveInlining)]
59          public static void InvokeWithModifiedMaxStackSize(Action action, int maxStackSize) =>
            ↪  StartNew(action, maxStackSize).Join();
60
61          /// <summary>
62          /// <para>Invokes the <see cref="Action"/> with extend maximum stack size.</para>
63          /// <para>Вызывает <see cref="Action"/> с расширенным максимальным размером стека.</para>
64          /// </summary>
65          /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делегат
            ↪  <see cref="Action"/>.</para></param>
66          [MethodImpl(MethodImplOptions.AggressiveInlining)]
67          public static void InvokeWithExtendedMaxStackSize(Action action) =>
            ↪  InvokeWithModifiedMaxStackSize(action, DefaultExtendedMaxStackSize);
68
69          /// <summary>
70          /// <para>Initializes a new instance of the <see cref="Thread"/> class, causes the
            ↪  operating system to change the state of that instance to <see
            ↪  cref="ThreadState.Running"/> and supplies an object containing data to be used by
            ↪  the method that thread executes.</para>
71          /// <para>Инициализирует новый экземпляр класса <see cref="Thread"/>, просит
            ↪  операционную систему изменить состояние этого экземпляра на <see
            ↪  cref="ThreadState.Running"/> и предоставляет объект, содержащий данные, которые
            ↪  будут использоваться в методе, который выполняет этот поток.</para>
72          /// </summary>
73          /// <typeparam name="T"><para>The type of the <see cref="Action{T}"/>
            ↪  argument.</para><para>Тип аргумента <see cref="Action{T}"/>.</para></typeparam>
74          /// <param name="param"><para>The object containing data to be used by the method that
            ↪  thread executes.</para><para>Объект, содержащий данные, которые будут использоваться
            ↪  методом, выполняемым потоком.</para></param>
75          /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делегат
            ↪  <see cref="Action{T}"/>.</para></param>
76          /// <param name="maxStackSize"><para>The maximum stack size in
            ↪  bytes.</para><para>Максимальный размер стека в байтах.</para></param>
77          /// <returns><para>A new started <see cref="Thread"/> instance.</para><para>Новый
            ↪  запущенный экземпляр <see cref="Thread"/>.</para></returns>
78          [MethodImpl(MethodImplOptions.AggressiveInlining)]
79          public static Thread StartNew<T>(T param, Action<object> action, int maxStackSize)
80          {
81              var thread = new Thread(new ParameterizedThreadStart(action), maxStackSize);
82              thread.Start(param);
83              return thread;
84          }
85
86          /// <summary>
87          /// <para>Initializes a new instance of the <see cref="Thread"/> class, causes the
            ↪  operating system to change the state of that instance to <see
            ↪  cref="ThreadState.Running"/> and supplies an object containing data to be used by
            ↪  the method that thread executes.</para>
88          /// <para>Инициализирует новый экземпляр класса <see cref="Thread"/>, просит
            ↪  операционную систему изменить состояние этого экземпляра на <see
            ↪  cref="ThreadState.Running"/> и предоставляет объект, содержащий данные, которые
            ↪  будут использоваться в методе, который выполняет этот поток.</para>
89          /// </summary>
90          /// <typeparam name="T"><para>The type of the <see cref="Action{T}"/>
            ↪  argument.</para><para>Тип аргумента <see cref="Action{T}"/>.</para></typeparam>
91          /// <param name="param"><para>The object containing data to be used by the method that
            ↪  thread executes.</para><para>Объект, содержащий данные, которые будут использоваться
            ↪  методом, выполняемым потоком.</para></param>
92          /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делегат
            ↪  <see cref="Action{T}"/>.</para></param>
93          /// <returns><para>A new started <see cref="Thread"/> instance.</para><para>Новый
            ↪  запущенный экземпляр <see cref="Thread"/>.</para></returns>
94          [MethodImpl(MethodImplOptions.AggressiveInlining)]
95          public static Thread StartNew<T>(T param, Action<object> action) => StartNew(param,
            ↪  action, DefaultMaxStackSize);
96
97          /// <summary>
```

```
98          /// <para>Initializes a new instance of the <see cref="Thread"/> class, causes the
            ↪  operating system to change the state of that instance to <see
            ↪  cref="ThreadState.Running"/> and supplies the method executed by that thread.</para>
99          /// <para>Инициализирует новый экземпляр класса <see cref="Thread"/>, просит
            ↪  операционную систему изменить состояние этого экземпляра на <see
            ↪  cref="ThreadState.Running"/> и предоставляет метод, который выполняется этим
            ↪  потоком.</para>
100         /// </summary>
101         /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делагат
            ↪  <see cref="Action"/>.</para></param>
102         /// <param name="maxStackSize"><para>The maximum stack size in
            ↪  bytes.</para><para>Максимальный размер стека в байтах.</para></param>
103         /// <returns><para>A new started <see cref="Thread"/> instance.</para><para>Новый
            ↪  запущенный экземпляр <see cref="Thread"/>.</para></returns>
104         [MethodImpl(MethodImplOptions.AggressiveInlining)]
105         public static Thread StartNew(Action action, int maxStackSize)
106         {
107             var thread = new Thread(new ThreadStart(action), maxStackSize);
108             thread.Start();
109             return thread;
110         }
111
112         /// <summary>
113         /// <para>Initializes a new instance of the <see cref="Thread"/> class, causes the
            ↪  operating system to change the state of that instance to <see
            ↪  cref="ThreadState.Running"/> and supplies the method executed by that thread.</para>
114         /// <para>Инициализирует новый экземпляр класса <see cref="Thread"/>, просит
            ↪  операционную систему изменить состояние этого экземпляра на <see
            ↪  cref="ThreadState.Running"/> и предоставляет метод, который выполняется этим
            ↪  потоком.</para>
115         /// </summary>
116         /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делагат
            ↪  <see cref="Action"/>.</para></param>
117         /// <returns><para>A new started <see cref="Thread"/> instance.</para><para>Новый
            ↪  запущенный экземпляр <see cref="Thread"/>.</para></returns>
118         [MethodImpl(MethodImplOptions.AggressiveInlining)]
119         public static Thread StartNew(Action action) => StartNew(action, DefaultMaxStackSize);
120
121         /// <summary>
122         /// Suspends the current thread for the <see cref="DefaultSleepInterval"/>.
123         /// </summary>
124         [MethodImpl(MethodImplOptions.AggressiveInlining)]
125         public static void Sleep() => Thread.Sleep(DefaultSleepInterval);
126     }
127 }
```

1.9 ./csharp/Platform.Threading.Tests/ThreadHelpersTests.cs

```
1  using Xunit;
2
3  namespace Platform.Threading.Tests
4  {
5      public class ThreadHelpersTests
6      {
7          [Fact]
8          public void InvokeTest()
9          {
10             var number = 0;
11             ThreadHelpers.InvokeWithExtendedMaxStackSize(() => number = 1);
12             Assert.Equal(1, number);
13             ThreadHelpers.InvokeWithExtendedMaxStackSize(2, (param) => number = (int)param);
14             Assert.Equal(2, number);
15             ThreadHelpers.InvokeWithModifiedMaxStackSize(() => number = 1, maxStackSize: 512);
16             Assert.Equal(1, number);
17             ThreadHelpers.InvokeWithModifiedMaxStackSize(2, (param) => number = (int)param,
               ↪  maxStackSize: 512);
18             Assert.Equal(2, number);
19         }
20     }
21 }
```

Index