

LinksPlatform's Platform.Threading Class Library

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

```
1 using System;
2 using System.Collections.Concurrent;
3 using System.Runtime.CompilerServices;
4 using System.Threading.Tasks;
5 using Platform.Collections.Concurrent;
6
7 namespace Platform.Threading
8 {
9     /// <summary>
10    /// <para>Provides a set of extension methods for <see cref="ConcurrentQueue{T}" />
11    ///   objects.</para>
12    /// <para>Предоставляет набор методов расширения для объектов <see
13    ///   cref="ConcurrentQueue{T}" />.</para>
14    /// </summary>
15    public static class ConcurrentQueueExtensions
16    {
17        /// <summary>
18        /// <para>Suspends evaluation of the method until all asynchronous operations in the
19        ///   queue finish.</para>
20        /// <para>Приостанавливает выполнение метода до завершения всех асинхронных операций в
21        ///   очереди.</para>
22        /// </summary>
23        /// <param name="queue"><para>The queue of asynchronous operations.</para><para>Очередь
24        ///   асинхронных операций.</para></param>
25        /// <returns><para>An asynchronous operation representation.</para><para>Представление
26        ///   асинхронной операции.</para></returns>
27        [MethodImpl(MethodImplOptions.AggressiveInlining)]
28        public static async Task AwaitAll(this ConcurrentQueue<Task> queue)
29        {
30            foreach (var item in queue.DequeueAll())
31            {
32                await item.ConfigureAwait(continueOnCapturedContext: false);
33            }
34        }
35
36        /// <summary>
37        /// <para>Suspends evaluation of the method until the first asynchronous operation in
38        ///   the queue finishes.</para>
39        /// <para>Приостанавливает выполнение метода до завершения первой асинхронной операции в
40        ///   очереди.</para>
41        /// </summary>
42        /// <param name="queue"><para>The queue of asynchronous operations.</para><para>Очередь
43        ///   асинхронных операций.</para></param>
44        /// <returns><para>An asynchronous operation representation.</para><para>Представление
45        ///   асинхронной операции.</para></returns>
46        [MethodImpl(MethodImplOptions.AggressiveInlining)]
47        public static async Task AwaitOne(this ConcurrentQueue<Task> queue)
48        {
49            if (queue.TryDequeue(out Task item))
50            {
51                await item.ConfigureAwait(continueOnCapturedContext: false);
52            }
53        }
54
55        /// <summary>
56        /// <para>Adds an <see cref="Action" /> as runned <see cref="Task" /> to the end of the
57        ///   <see cref="ConcurrentQueue{T}" />.</para>
58        /// <para>Добавляет <see cref="Action" /> как запущенную <see cref="Task" /> в конец <see
59        ///   cref="ConcurrentQueue{T}" />.</para>
60        /// </summary>
61        /// <param name="queue"><para>The queue of asynchronous operations.</para><para>Очередь
62        ///   асинхронных операций.</para></param>
63        /// <param name="action"><para>The <see cref="Action" /> delegate.</para><para>Делегат
64        ///   <see cref="Action" />.</para></param>
65        [MethodImpl(MethodImplOptions.AggressiveInlining)]
66        public static void EnqueueAsRunnedTask(this ConcurrentQueue<Task> queue, Action action)
67        {
68            queue.Enqueue(Task.Run(action));
69        }
70    }
71 }
```

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

```
1 using System;
2 using System.Runtime.CompilerServices;
3
4 namespace Platform.Threading.Synchronization
```

```

5 {
6     /// <summary>
7     /// <para>Represents a synchronization object that supports read and write operations.</para>
8     /// <para>Представляет объект синхронизации с поддержкой операций чтения и записи.</para>
9     /// </summary>
10    public interface ISynchronization
11    {
12        /// <summary>
13        /// <para>Executes action in read access mode.</para>
14        /// <para>Выполняет действие в режиме доступа для чтения.</para>
15        /// </summary>
16        /// <param name="action"><para>The action.</para><para>Действие.</para></param>
17        [MethodImpl(MethodImplOptions.AggressiveInlining)]
18        void ExecuteReadOperation(Action action);
19
20        /// <summary>
21        /// <para>Executes a function in read access mode and returns the function's
22        → result.</para>
23        /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё
24        → результат.</para>
25        /// </summary>
26        /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
27        → результата функции.</para></typeparam>
28        /// <param name="function"><para>The function.</para><para>Функция.</para></param>
29        /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
30        [MethodImpl(MethodImplOptions.AggressiveInlining)]
31        TResult ExecuteReadOperation<TResult>(Func<TResult> function);
32
33        /// <summary>
34        /// <para>Executes action in write access mode.</para>
35        /// <para>Выполняет действие в режиме доступа для записи.</para>
36        /// </summary>
37        /// <param name="action"><para>The action.</para><para>Действие.</para></param>
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        void ExecuteWriteOperation(Action action);
40
41        /// <summary>
42        /// <para>Executes a function in write access mode and returns the function's
43        → result.</para>
44        /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
45        → результат.</para>
46        /// </summary>
47        /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
48        → результата функции.</para></typeparam>
49        /// <param name="function"><para>The function.</para><para>Функция.</para></param>
50        /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
51        [MethodImpl(MethodImplOptions.AggressiveInlining)]
52        TResult ExecuteWriteOperation<TResult>(Func<TResult> function);
53    }
54 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3
4  namespace Platform.Threading.Synchronization
5  {
6      /// <summary>
7      /// <para>Contains extension methods for the <see cref="ISynchronization"/> interface.</para>
8      /// <para>Содержит методы расширения для интерфейса <see cref="ISynchronization"/>.</para>
9      /// </summary>
10     public static class ISynchronizationExtensions
11     {
12         /// <summary>
13         /// <para>Executes a function in read access mode and returns the function's
14         → result.</para>
15         /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё
16         → результат.</para>
17         /// </summary>
18         /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
19         → результата функции.</para></typeparam>
20         /// <typeparam name="TParam"><para>The parameter type.</para><para>Тип
21         → параметра.</para></typeparam>
22         /// <param name="synchronization"><para>Synchronization
23         → object.</para><para>Синхронизация объекта.</para></param>
24         /// <param name="parameter"><para>The parameter</para><para>Параметр.</para></param>
25         /// <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 ExecuteReadOperation<TResult, TParam>(this ISynchronization
    → synchronization, TParam parameter, Func<TParam, TResult> function) =>
    → synchronization.ExecuteReadOperation(() => 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 ExecuteReadOperation<TParam>(this ISynchronization synchronization,
    → TParam parameter, Action<TParam> action) => synchronization.ExecuteReadOperation(()
    → => 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 ExecuteWriteOperation<TResult, TParam>(this ISynchronization
    → synchronization, TParam parameter, Func<TParam, TResult> function) =>
    → synchronization.ExecuteWriteOperation(() => 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 ExecuteWriteOperation<TParam>(this ISynchronization synchronization,
    → TParam parameter, Action<TParam> action) => synchronization.ExecuteWriteOperation(()
    → => 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>

```

```

[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static TResult ExecuteReadOperation<TResult, TParam1, TParam2>(this
    ↳ ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2,
    ↳ Func<TParam1, TParam2, TResult> function) => synchronization.ExecuteReadOperation(()
    ↳ => function(parameter1, parameter2));

/// <summary>
/// <para>Executes action in read access mode.</para>
/// <para>Выполняет действие в режиме доступа для чтения.</para>
/// </summary>
/// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
    ↳ параметра.</para></typeparam>
/// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
    ↳ параметра.</para></typeparam>
/// <param name="synchronization"><para>Synchronization
    ↳ object.</para><para>Синхронизация объекта.</para></param>
/// <param name="parameter1"><para>The first parameter</para><para>Первый
    ↳ параметр.</para></param>
/// <param name="parameter2"><para>The second parameter</para><para>Второй
    ↳ параметр.</para></param>
/// <param name="action"><para>The action.</para><para>Действие.</para></param>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static void ExecuteReadOperation<TParam1, TParam2>(this ISynchronization
    ↳ synchronization, TParam1 parameter1, TParam2 parameter2, Action<TParam1, TParam2>
    ↳ action) => synchronization.ExecuteReadOperation(() => action(parameter1,
    ↳ parameter2));

/// <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>
/// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
    ↳ параметра.</para></typeparam>
/// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
    ↳ параметра.</para></typeparam>
/// <param name="synchronization"><para>Synchronization
    ↳ object.</para><para>Синхронизация объекта.</para></param>
/// <param name="parameter1"><para>The first parameter</para><para>Первый
    ↳ параметр.</para></param>
/// <param name="parameter2"><para>The second parameter</para><para>Второй
    ↳ параметр.</para></param>
/// <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)]
public static TResult ExecuteWriteOperation<TResult, TParam1, TParam2>(this
    ↳ ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2,
    ↳ Func<TParam1, TParam2, TResult> function) =>
    ↳ synchronization.ExecuteWriteOperation(() => function(parameter1, parameter2));

/// <summary>
/// <para>Executes action in write access mode.</para>
/// <para>Выполняет действие в режиме доступа для записи.</para>
/// </summary>
/// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
    ↳ параметра.</para></typeparam>
/// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
    ↳ параметра.</para></typeparam>
/// <param name="synchronization"><para>Synchronization
    ↳ object.</para><para>Синхронизация объекта.</para></param>
/// <param name="parameter1"><para>The first parameter</para><para>Первый
    ↳ параметр.</para></param>
/// <param name="parameter2"><para>The second parameter</para><para>Второй
    ↳ параметр.</para></param>
/// <param name="action"><para>The action.</para><para>Действие.</para></param>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static void ExecuteWriteOperation<TParam1, TParam2>(this ISynchronization
    ↳ synchronization, TParam1 parameter1, TParam2 parameter2, Action<TParam1, TParam2>
    ↳ action) => synchronization.ExecuteWriteOperation(() => action(parameter1,
    ↳ parameter2));

/// <summary>

```

```

117 /// <para>Executes a function in read access mode and returns the function's
118   → result.</para>
119 /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё
120   → результат.</para>
121 /// </summary>
122 /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
123   → результата функции.</para></typeparam>
124 /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
125   → параметра.</para></typeparam>
126 /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
127   → параметра.</para></typeparam>
128 /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
129   → параметра.</para></typeparam>
130 /// <param name="synchronization"><para>Synchronization
131   → object.</para><para>Синхронизация объекта.</para></param>
132 /// <param name="parameter1"><para>The first parameter</para><para>Первый
133   → параметр.</para></param>
134 /// <param name="parameter2"><para>The second parameter</para><para>Второй
135   → параметр.</para></param>
136 /// <param name="parameter3"><para>The third parameter</para><para>Третий
137   → параметр.</para></param>
138 /// <param name="function"><para>The function.</para><para>Функция.</para></param>
139 /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 public static TResult ExecuteReadOperation<TResult, TParam1, TParam2, TParam3>(this
142   → ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2, TParam3
143   → parameter3, Func<TParam1, TParam2, TParam3, TResult> function) =>
144   → synchronization.ExecuteReadOperation(() => function(parameter1, parameter2,
145   → parameter3));
146
147 /// <summary>
148 /// <para>Executes action in read access mode.</para>
149 /// <para>Выполняет действие в режиме доступа для чтения.</para>
150 /// </summary>
151 /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
152   → параметра.</para></typeparam>
153 /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
154   → параметра.</para></typeparam>
155 /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
156   → параметра.</para></typeparam>
157 /// <param name="synchronization"><para>Synchronization
158   → object.</para><para>Синхронизация объекта.</para></param>
159 /// <param name="parameter1"><para>The first parameter</para><para>Первый
160   → параметр.</para></param>
161 /// <param name="parameter2"><para>The second parameter</para><para>Второй
162   → параметр.</para></param>
163 /// <param name="parameter3"><para>The third parameter</para><para>Третий
164   → параметр.</para></param>
165 /// <param name="action"><para>The action.</para><para>Действие.</para></param>
166 [MethodImpl(MethodImplOptions.AggressiveInlining)]
167 public static void ExecuteReadOperation<TParam1, TParam2, TParam3>(this ISynchronization
168   → synchronization, TParam1 parameter1, TParam2 parameter2, TParam3 parameter3,
169   → Action<TParam1, TParam2, TParam3> action) => synchronization.ExecuteReadOperation(()
170   → => action(parameter1, parameter2, parameter3));
171
172 /// <summary>
173 /// <para>Executes a function in write access mode and returns the function's
174   → result.</para>
175 /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
176   → результат.</para>
177 /// </summary>
178 /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
179   → результата функции.</para></typeparam>
180 /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
181   → параметра.</para></typeparam>
182 /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
183   → параметра.</para></typeparam>
184 /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
185   → параметра.</para></typeparam>
186 /// <param name="synchronization"><para>Synchronization
187   → object.</para><para>Синхронизация объекта.</para></param>
188 /// <param name="parameter1"><para>The first parameter</para><para>Первый
189   → параметр.</para></param>
190 /// <param name="parameter2"><para>The second parameter</para><para>Второй
191   → 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 ExecuteWriteOperation<TResult, TParam1, TParam2, TParam3>(this
    ↳ ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2, TParam3
    ↳ parameter3, Func<TParam1, TParam2, TParam3, TResult> function) =>
    ↳ synchronization.ExecuteWriteOperation(() => 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 ExecuteWriteOperation<TParam1, TParam2, TParam3>(this
    ↳ ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2, TParam3
    ↳ parameter3, Action<TParam1, TParam2, TParam3> action) =>
    ↳ synchronization.ExecuteWriteOperation(() => 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>Четвёртый
    ↳ 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 ExecuteReadOperation<TResult, TParam1, TParam2, TParam3,
    ↳ TParam4>(this ISynchronization synchronization, TParam1 parameter1, TParam2
    ↳ parameter2, TParam3 parameter3, TParam4 parameter4, Func<TParam1, TParam2, TParam3,
    ↳ TParam4, TResult> function) => synchronization.ExecuteReadOperation(() =>
    ↳ 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
204   ↳ параметра.</para></typeparam>
205 /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
206   ↳ параметра.</para></typeparam>
207 /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
208   ↳ параметра.</para></typeparam>
209 /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
210   ↳ параметра.</para></typeparam>
211 /// <param name="synchronization"><para>Synchronization
212   ↳ object.</para><para>Синхронизация объекта.</para></param>
213 /// <param name="parameter1"><para>The first parameter</para><para>Первый
214   ↳ параметр.</para></param>
215 /// <param name="parameter2"><para>The second parameter</para><para>Второй
216   ↳ параметр.</para></param>
217 /// <param name="parameter3"><para>The third parameter</para><para>Третий
218   ↳ параметр.</para></param>
219 /// <param name="parameter4"><para>The forth parameter</para><para>Четвёртый
220   ↳ параметр.</para></param>
221 /// <param name="action"><para>The action.</para><para>Действие.</para></param>
222 [MethodImpl(MethodImplOptions.AggressiveInlining)]
223 public static void ExecuteReadOperation<TParam1, TParam2, TParam3, TParam4>(this
224   ↳ ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2, TParam3
225   ↳ parameter3, TParam4 parameter4, Action<TParam1, TParam2, TParam3, TParam4> action)
226   ↳ => synchronization.ExecuteReadOperation(() => action(parameter1, parameter2,
227   ↳ parameter3, parameter4));
228
229 /// <summary>
230 /// <para>Executes a function in write access mode and returns the function's
231   ↳ result.</para>
232 /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
233   ↳ результат.</para>
234 /// </summary>
235 /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
236   ↳ результата функции.</para></typeparam>
237 /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
238   ↳ параметра.</para></typeparam>
239 /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
240   ↳ параметра.</para></typeparam>
241 /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
242   ↳ параметра.</para></typeparam>
243 /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
244   ↳ параметра.</para></typeparam>
245 /// <param name="synchronization"><para>Synchronization
246   ↳ object.</para><para>Синхронизация объекта.</para></param>
247 /// <param name="parameter1"><para>The first parameter</para><para>Первый
248   ↳ параметр.</para></param>
249 /// <param name="parameter2"><para>The second parameter</para><para>Второй
250   ↳ параметр.</para></param>
251 /// <param name="parameter3"><para>The third parameter</para><para>Третий
252   ↳ параметр.</para></param>
253 /// <param name="parameter4"><para>The forth parameter</para><para>Четвёртый
254   ↳ параметр.</para></param>
255 /// <param name="function"><para>The function.</para><para>Функция.</para></param>
256 /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
257 [MethodImpl(MethodImplOptions.AggressiveInlining)]
258 public static TResult ExecuteWriteOperation<TResult, TParam1, TParam2, TParam3,
259   ↳ TParam4>(this ISynchronization synchronization, TParam1 parameter1, TParam2
260   ↳ parameter2, TParam3 parameter3, TParam4 parameter4, Func<TParam1, TParam2, TParam3,
261   ↳ TParam4, TResult> function) => synchronization.ExecuteWriteOperation(() =>
262   ↳ function(parameter1, parameter2, parameter3, parameter4));
263
264 /// <summary>
265 /// <para>Executes action in write access mode.</para>
266 /// <para>Выполняет действие в режиме доступа для записи.</para>
267 /// </summary>
268 /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
269   ↳ параметра.</para></typeparam>
270 /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
271   ↳ параметра.</para></typeparam>
272 /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
273   ↳ параметра.</para></typeparam>
274 /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
275   ↳ параметра.</para></typeparam>
276 /// <param name="synchronization"><para>Synchronization
277   ↳ 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 ExecuteWriteOperation(TParam1, TParam2, TParam3, TParam4>(this
    ↪ ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2, TParam3
    ↪ parameter3, TParam4 parameter4, Action<TParam1, TParam2, TParam3, TParam4> action)
    ↪ => synchronization.ExecuteWriteOperation(() => 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
    ↪ guarantee 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
    ↪ guarantee 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
9     ↪ cref="ReaderWriterLockSlim"/>.</para>
10    /// <para>Реализация <see cref="ISynchronization"/> на основе <see
11    ↪ cref="ReaderWriterLockSlim"/>.</para>
12    /// </summary>
13    public class ReaderWriterLockSynchronization : ISynchronization
14    {
15        private readonly ReaderWriterLockSlim _rwLock = new
16            ↪ ReaderWriterLockSlim(LockRecursionPolicy.SupportsRecursion);
17
18        /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
19        ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
20        ↪ n.ExecuteReadOperation(System.Action)"]/*' />
21        /// <inheritdoc/>
22        [MethodImpl(MethodImplOptions.AggressiveInlining)]
23        public void ExecuteReadOperation(Action action)
24        {
25            _rwLock.EnterReadLock();
26            try
27            {
28                action();
29            }
30            finally
31            {
32                _rwLock.ExitReadLock();
33            }
34        }
35
36        /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
37        ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
38        ↪ n.ExecuteReadOperation`1(System.Func{`0})"]/*' />
39        /// <inheritdoc/>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]
41        public TResult ExecuteReadOperation<TResult>(Func<TResult> function)
42        {
43            _rwLock.EnterReadLock();
44            try
45            {
46                return function();
47            }
48            finally
49            {
50                _rwLock.ExitReadLock();
51            }
52        }
53
54        /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
55        ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
56        ↪ n.ExecuteWriteOperation(System.Action)"]/*' />
57        /// <inheritdoc/>
58        [MethodImpl(MethodImplOptions.AggressiveInlining)]
59        public void ExecuteWriteOperation(Action action)
60        {
61            _rwLock.EnterWriteLock();
62            try
63            {
64                action();
65            }
66            finally
67            {
68                _rwLock.ExitWriteLock();
69            }
70        }
71
72        /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
73        ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
74        ↪ n.ExecuteWriteOperation`1(System.Func{`0})"]/*' />
75        /// <inheritdoc/>
```

```

65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     public TResult ExecuteWriteOperation<TResult>(Func<TResult> function)
67     {
68         _rwLock.EnterWriteLock();
69         try
70         {
71             return function();
72         }
73         finally
74         {
75             _rwLock.ExitWriteLock();
76         }
77     }
78 }
79 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3
4  namespace Platform.Threading.Synchronization
5  {
6      /// <summary>
7      /// <para>Implementation of <see cref="ISynchronization"/> that makes no actual
8      /// <para>Реализация <see cref="ISynchronization"/>, которая не выполняет фактическую
9      /// <para>синхронизацию.</para>
10     /// </summary>
11     public class Unsynchronization : ISynchronization
12     {
13         /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
14         /// <para>path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronization.
15         /// <para>n.ExecuteReadOperation(System.Action)"]/*' />
16         /// <inheritdoc/>
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public void ExecuteReadOperation(Action action) => action();
19
20         /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
21         /// <para>path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronization.
22         /// <para>n.ExecuteReadOperation`1(System.Func{`0})"]/*' />
23         /// <inheritdoc/>
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         public TResult ExecuteReadOperation<TResult>(Func<TResult> function) => function();
26
27         /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
28         /// <para>path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronization.
29         /// <para>n.ExecuteWriteOperation(System.Action)"]/*' />
30         /// <inheritdoc/>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public void ExecuteWriteOperation(Action action) => action();
33
34         /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
35         /// <para>path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronization.
36         /// <para>n.ExecuteWriteOperation`1(System.Func{`0})"]/*' />
37         /// <inheritdoc/>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public TResult ExecuteWriteOperation<TResult>(Func<TResult> function) => function();
40     }
41 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using System.Threading.Tasks;
3
4  namespace Platform.Threading
5  {
6      /// <summary>
7      /// <para>Provides a set of extension methods for <see cref="Task{TResult}"/> objects.</para>
8      /// <para>Предоставляет набор методов расширения для объектов <see
9      /// <para>cref="Task{TResult}"/>.</para>
10     /// </summary>
11     public static class TaskExtensions
12     {
13         /// <summary>
14         /// <para>Waits for completion of the asynchronous <see cref="Task{TResult}"/> and
15         /// <para>returns its result.</para>
16         /// <para>Ожидает завершения асинхронной <see cref="Task{TResult}"/> и возвращает её
17         /// <para>результат.</para>
18     }
19 }

```

```

15     /// </summary>
16     /// <typeparam name="TReturn"><para>The return value type.</para><para>Тип возвращаемого
    → значения.</para></typeparam>
17     /// <param name="task"><para>The asynchronous <see
    → cref="Task{TReturn}">.</para><para>Асинхронная <see
    → cref="Task{TReturn}">.</para></param>
18     /// <returns><para>The result of completed <see
    → cref="Task{TReturn}">.</para><para>Результат завершённой <see
    → cref="Task{TReturn}">.</para></returns>
19     [MethodImpl(MethodImplOptions.AggressiveInlining)]
20     public static TReturn AwaitResult<TReturn>(this Task<TReturn> task) =>
    → task.GetAwaiter().GetResult();
21 }
22 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using System.Threading;
4
5  namespace Platform.Threading
6  {
7      /// <summary>
8      /// <para>Provides a set of helper methods for <see cref="Thread"/> objects.</para>
9      /// <para>Предоставляет набор вспомогательных методов для объектов <see
    → cref="Thread"/>.</para>
10     /// </summary>
11     public static class ThreadHelpers
12     {
13         /// <summary>
14         /// <para>Gets the maximum stack size in bytes by default.</para>
15         /// <para>Возвращает размер максимальный стека в байтах по умолчанию.</para>
16         /// </summary>
17         public static readonly int DefaultMaxStackSize;
18
19         /// <summary>
20         /// <para>Gets the extended maximum stack size in bytes by default.</para>
21         /// <para>Возвращает расширенный максимальный размер стека в байтах по умолчанию.</para>
22         /// </summary>
23         public static readonly int DefaultExtendedMaxStackSize = 256 * 1024 * 1024;
24
25         /// <summary>
26         /// <para>Returns the default time interval for transferring control to other threads in
    → milliseconds</para>
27         /// <para>Возвращает интервал времени для передачи управления другим потокам в
    → миллисекундах по умолчанию.</para>
28         /// </summary>
29         public static readonly int DefaultSleepInterval = 1;
30
31         /// <summary>
32         /// <para>Invokes the <see cref="Action{T}"> with modified maximum stack size.</para>
33         /// <para>Вызывает <see cref="Action{T}"> с изменённым максимальным размером
    → стека.</para>
34         /// </summary>
35         /// <typeparam name="T"><para>The type of the <see cref="Action{T}">
    → argument.</para><para>Тип аргумента <see cref="Action{T}">.</para></typeparam>
36         /// <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 вызываемым делегатом <see cref="Action{T}">.</para></param>
37         /// <param name="action"><para>The <see cref="Action{T}"> delegate.</para><para>Делегат
    → <see cref="Action{T}">.</para></param>
38         /// <param name="maxStackSize"><para>The maximum stack size in
    → bytes.</para><para>Максимальный размер стека в байтах.</para></param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public static void InvokeWithModifiedMaxStackSize<T>(T param, Action<object> action, int
    → maxStackSize) => StartNew(param, action, maxStackSize).Join();
41
42         /// <summary>
43         /// <para>Invokes the <see cref="Action{T}"> with extend maximum stack size.</para>
44         /// <para>Вызывает <see cref="Action{T}"> с расширенным максимальным размером
    → стека.</para>
45         /// </summary>
46         /// <typeparam name="T"><para>The type of the <see cref="Action{T}">
    → argument.</para><para>Тип аргумента <see cref="Action{T}">.</para></typeparam>
47         /// <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 вызываемым делегатом <see cref="Action{T}">.</para></param>

```

```

48  /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делает
    ↳ <see cref="Action{T}"/>.</para></param>
49  [MethodImpl(MethodImplOptions.AggressiveInlining)]
50  public static void InvokeWithExtendedMaxStackSize<T>(T param, Action<object> action) =>
    ↳ InvokeWithModifiedMaxStackSize(param, action, DefaultExtendedMaxStackSize);
51
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>Делает
    ↳ <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>Делает
    ↳ <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>Делает
    ↳ <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>Делает
    ↳ <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>Делегат
    ↳ <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>Делегат
    ↳ <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

- ./csharp/Platform.Threading.Tests/ThreadHelpersTests.cs, 13
- ./csharp/Platform.Threading/ConcurrentQueueExtensions.cs, 1
- ./csharp/Platform.Threading/Synchronization/ISynchronization.cs, 1
- ./csharp/Platform.Threading/Synchronization/ISynchronizationExtensions.cs, 2
- ./csharp/Platform.Threading/Synchronization/ISynchronized.cs, 8
- ./csharp/Platform.Threading/Synchronization/ReaderWriterLockSynchronization.cs, 9
- ./csharp/Platform.Threading/Synchronization/Unsynchronization.cs, 10
- ./csharp/Platform.Threading/TaskExtensions.cs, 10
- ./csharp/Platform.Threading/ThreadHelpers.cs, 11