LinksPlatform's Platform.Threading Class Library

./ConcurrentQueueExtensions.cs

```
using System;
using System.Collections.Concurrent;
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>
        public static async Task AwaitAll(this ConcurrentQueue<Task> queue)
        {
            foreach (var item in queue.DequeueAll())
            {
                await item;
            }
        }

        /// <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>
        public static async Task AwaitOne(this ConcurrentQueue<Task> queue)
        {
            if (queue.TryDequeue(out Task item))
            {
                await item;
            }
        }

        /// <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>
        public static void EnqueueAsRunnedTask(this ConcurrentQueue<Task> queue, Action action) => queue.Enqueue(Task.Run(action));
    }
}
```

./Synchronization/ISynchronization.cs

```
using System;

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>
        void ExecuteReadOperation(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>
        TResult ExecuteReadOperation<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>
        void ExecuteWriteOperation(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>
        TResult ExecuteWriteOperation<TResult>(Func<TResult> function);
    }
}
```

./Synchronization/ISynchronizationExtensions.cs

```
using System;

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>
        /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
        public static TResult ExecuteReadOperation<TResult, TParam>(this ISynchronization synchronization, TParam parameter, Func<TParam, TResult> function) => synchronization.ExecuteReadOperation(() => function(parameter));

        /// <summary>
        /// <para>Executes action in read access mode.</para>
        /// <para>Выполняет действие в режиме доступа для чтения.</para>
        /// </summary>
```

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

```
75          /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
76          /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
77          /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
78          /// <param name="action"><para>The action.</para><para>Действие.</para></param>
79          public static void ExecuteReadOperation<TParam1, TParam2>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, Action<TParam1, TParam2>
            ↪  action) => synchronization.ExecuteReadOperation(() => action(parameter1,
            ↪  parameter2));
80
81          /// <summary>
82          /// <para>Executes a function in write access mode and returns the function's
            ↪  result.</para>
83          /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
            ↪  результат.</para>
84          /// </summary>
85          /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
86          /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
87          /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
88          /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
89          /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
90          /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
91          /// <param name="function"><para>The function.</para><para>Функция.</para></param>
92          /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
93          public static TResult ExecuteWriteOperation<TResult, TParam1, TParam2>(this
            ↪  ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2,
            ↪  Func<TParam1, TParam2, TResult> function) =>
            ↪  synchronization.ExecuteWriteOperation(() => function(parameter1, parameter2));
94
95          /// <summary>
96          /// <para>Executes action in write access mode.</para>
97          /// <para>Выполняет действие в режиме доступа для записи.</para>
98          /// </summary>
99          /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
100         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
101         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
102         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
103         /// <param name="parameter2"><para>The second parameter</para><para>Второй
            ↪  параметр.</para></param>
104         /// <param name="action"><para>The action.</para><para>Действие.</para></param>
105         public static void ExecuteWriteOperation<TParam1, TParam2>(this ISynchronization
            ↪  synchronization, TParam1 parameter1, TParam2 parameter2, Action<TParam1, TParam2>
            ↪  action) => synchronization.ExecuteWriteOperation(() => action(parameter1,
            ↪  parameter2));
106
107         /// <summary>
108         /// <para>Executes a function in read access mode and returns the function's
            ↪  result.</para>
109         /// <para>Выполняет функцию в режиме доступа для чтения и возвращает полученный из неё
            ↪  результат.</para>
110         /// </summary>
111         /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
            ↪  результата функции.</para></typeparam>
112         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
            ↪  параметра.</para></typeparam>
113         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
            ↪  параметра.</para></typeparam>
114         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
            ↪  параметра.</para></typeparam>
115         /// <param name="synchronization"><para>Synchronization
            ↪  object.</para><para>Синхронизация объекта.</para></param>
116         /// <param name="parameter1"><para>The first parameter</para><para>Первый
            ↪  параметр.</para></param>
```

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

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

```
199         public static void ExecuteReadOperation<TParam1, TParam2, TParam3, TParam4>(this
                ISynchronization synchronization, TParam1 parameter1, TParam2 parameter2, TParam3
                parameter3, TParam4 parameter4, Action<TParam1, TParam2, TParam3, TParam4> action)
                => synchronization.ExecuteReadOperation(() => action(parameter1, parameter2,
                parameter3, parameter4));
200
201         /// <summary>
202         /// <para>Executes a function in write access mode and returns the function's
                result.</para>
203         /// <para>Выполняет функцию в режиме доступа для записи и возвращает полученный из неё
                результат.</para>
204         /// </summary>
205         /// <typeparam name="TResult"><para>Type of function's result.</para><para>Тип
                результата функции.</para></typeparam>
206         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
                параметра.</para></typeparam>
207         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
                параметра.</para></typeparam>
208         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
                параметра.</para></typeparam>
209         /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
                параметра.</para></typeparam>
210         /// <param name="synchronization"><para>Synchronization
                object.</para><para>Синхронизация объекта.</para></param>
211         /// <param name="parameter1"><para>The first parameter</para><para>Первый
                параметр.</para></param>
212         /// <param name="parameter2"><para>The second parameter</para><para>Второй
                параметр.</para></param>
213         /// <param name="parameter3"><para>The third parameter</para><para>Третий
                параметр.</para></param>
214         /// <param name="parameter4"><para>The forth parameter</para><para>Чертвёртый
                параметр.</para></param>
215         /// <param name="function"><para>The function.</para><para>Функция.</para></param>
216         /// <returns><para>The function's result.</para><para>Результат функции.</para></returns>
217         public static TResult ExecuteWriteOperation<TResult, TParam1, TParam2, TParam3,
                TParam4>(this ISynchronization synchronization, TParam1 parameter1, TParam2
                parameter2, TParam3 parameter3, TParam4 parameter4, Func<TParam1, TParam2, TParam3,
                TParam4, TResult> function) => synchronization.ExecuteWriteOperation(() =>
                function(parameter1, parameter2, parameter3, parameter4));
218
219         /// <summary>
220         /// <para>Executes action in write access mode.</para>
221         /// <para>Выполняет действие в режиме доступа для записи.</para>
222         /// </summary>
223         /// <typeparam name="TParam1"><para>The first parameter type.</para><para>Тип первого
                параметра.</para></typeparam>
224         /// <typeparam name="TParam2"><para>The second parameter type.</para><para>Тип второго
                параметра.</para></typeparam>
225         /// <typeparam name="TParam3"><para>The third parameter type.</para><para>Тип третьего
                параметра.</para></typeparam>
226         /// <typeparam name="TParam4"><para>The forth parameter type.</para><para>Тип четвёртого
                параметра.</para></typeparam>
227         /// <param name="synchronization"><para>Synchronization
                object.</para><para>Синхронизация объекта.</para></param>
228         /// <param name="parameter1"><para>The first parameter</para><para>Первый
                параметр.</para></param>
229         /// <param name="parameter2"><para>The second parameter</para><para>Второй
                параметр.</para></param>
230         /// <param name="parameter3"><para>The third parameter</para><para>Третий
                параметр.</para></param>
231         /// <param name="parameter4"><para>The forth parameter</para><para>Чертвёртый
                параметр.</para></param>
232         /// <param name="action"><para>The action.</para><para>Действие.</para></param>
233         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));
234     }
235 }
```

./Synchronization/ISynchronized.cs

```
1  namespace Platform.Threading.Synchronization
2  {
3      /// <summary>
4      /// <para>Represents extendable synchronized interface access gate.</para>
```

```
5       /// <para>Представляет расширяемый интерфейс шлюза синхронизированного доступа.</para>
6       /// </summary>
7       /// <typeparam name="TInterface"><para>Synchronized interface.</para><para>Синхронизируемый
        ↪  интерфейс.</para></typeparam>
8       public interface ISynchronized<out TInterface>
9       {
10          /// <summary>
11          /// <para>Gets sychronization method.</para>
12          /// <para>Возвращает способ синхронизации.</para>
13          /// </summary>
14          ISynchronization SyncRoot { get; }
15
16          /// <summary>
17          /// <para>Get source version of <typeparamref name="TInterface"/>, that does not
            ↪  garantee thread safe access synchronization.</para>
18          /// <para>Возвращает исходную версию <typeparamref name="TInterface"/>, которая не
            ↪  гарантирует потокобезопасную синхронизацию доступа.</para>
19          /// </summary>
20          /// <remarks>
21          /// <para>It is unsafe to use it directly, unless compound context using SyncRoot is
            ↪  created.</para>
22          /// <para>Использовать напрямую небезопасно, за исключением ситуации когда создаётся
            ↪  составной контекст с использованием SyncRoot.</para>
23          /// </remarks>
24          TInterface Unsync { get; }
25
26          /// <summary>
27          /// <para>Get wrapped/decorated version of <typeparamref name="TInterface"/>, that does
            ↪  garantee thread safe access synchronization.</para>
28          /// <para>Возвращает обернутую/декорированную версию <typeparamref name="TInterface"/>,
            ↪  которая гарантирует потокобезопасную синхронизацию доступа.</para>
29          /// </summary>
30          /// <remarks>
31          /// <para>It is safe to use it directly, because it must be thread safe
            ↪  implementation.</para>
32          /// <para>Безопасно использовать напрямую, так как реализация должна быть
            ↪  потокобезопасной.</para>
33          /// </remarks>
34          TInterface Sync { get; }
35      }
36  }
```

./Synchronization/ReaderWriterLockSynchronization.cs

```
1   using System;
2   using System.Threading;
3
4   namespace Platform.Threading.Synchronization
5   {
6       /// <summary>
7       /// <para>Implementation of <see cref="ISynchronization"/> based on <see
        ↪  cref="ReaderWriterLockSlim"/>.</para>
8       /// <para>Реализация <see cref="ISynchronization"/> на основе <see
        ↪  cref="ReaderWriterLockSlim"/>.</para>
9       /// </summary>
10      public class ReaderWriterLockSynchronization : ISynchronization
11      {
12          private readonly ReaderWriterLockSlim _rwLock = new
            ↪  ReaderWriterLockSlim(LockRecursionPolicy.SupportsRecursion);
13
14          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪  path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪  n.ExecuteReadOperation(System.Action)"]/*'/>
15          /// <inheritdoc/>
16          public void ExecuteReadOperation(Action action)
17          {
18              _rwLock.EnterReadLock();
19              try
20              {
21                  action();
22              }
23              finally
24              {
25                  _rwLock.ExitReadLock();
26              }
27          }
28
```

```
29          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪ n.ExecuteReadOperation``1(System.Func{``0})"]/*'/>
30          /// <inheritdoc/>
31          public TResult ExecuteReadOperation<TResult>(Func<TResult> function)
32          {
33              _rwLock.EnterReadLock();
34              try
35              {
36                  return function();
37              }
38              finally
39              {
40                  _rwLock.ExitReadLock();
41              }
42          }
43
44          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪ n.ExecuteWriteOperation(System.Action)"]/*'/>
45          /// <inheritdoc/>
46          public void ExecuteWriteOperation(Action action)
47          {
48              _rwLock.EnterWriteLock();
49              try
50              {
51                  action();
52              }
53              finally
54              {
55                  _rwLock.ExitWriteLock();
56              }
57          }
58
59          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪ n.ExecuteWriteOperation``1(System.Func{``0})"]/*'/>
60          /// <inheritdoc/>
61          public TResult ExecuteWriteOperation<TResult>(Func<TResult> function)
62          {
63              _rwLock.EnterWriteLock();
64              try
65              {
66                  return function();
67              }
68              finally
69              {
70                  _rwLock.ExitWriteLock();
71              }
72          }
73      }
74  }
```

./Synchronization/Unsynchronization.cs

```
1   using System;
2
3   namespace Platform.Threading.Synchronization
4   {
5       /// <summary>
6       /// <para>Implementation of <see cref="ISynchronization"/> that makes no actual
            ↪ synchronization.</para>
7       /// <para>Реализация <see cref="ISynchronization"/>, которая не выполняет фактическую
            ↪ синхронизацию.</para>
8       /// </summary>
9       public class Unsynchronization : ISynchronization
10      {
11          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪ n.ExecuteReadOperation(System.Action)"]/*'/>
12          /// <inheritdoc/>
13          public void ExecuteReadOperation(Action action) => action();
14
15          /// <include file='bin\Release\netstandard2.0\Platform.Threading.xml'
            ↪ path='doc/members/member[@name="M:Platform.Threading.Synchronization.ISynchronizatio
            ↪ n.ExecuteReadOperation``1(System.Func{``0})"]/*'/>
16          /// <inheritdoc/>
17          public TResult ExecuteReadOperation<TResult>(Func<TResult> function) => function();
18
```

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

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

./TaskExtensions.cs

```
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>
        public static TReturn AwaitResult<TReturn>(this Task<TReturn> task) => task.GetAwaiter().GetResult();
    }
}
```

./ThreadHelpers.cs

```
using System;
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 = 200 * 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;
```

```
29
30          /// <summary>
31          /// <para>Invokes the <see cref="Action{T}"/> with modified maximum stack size.</para>
32          /// <para>Вызывает <see cref="Action{T}"/> с изменённым максимальным размером
            ↪  стека.</para>
33          /// </summary>
34          /// <typeparam name="T"><para>The type of the <see cref="Action{T}"/>
            ↪  argument.</para><para>Тип аргумента <see cref="Action{T}"/>.</para></typeparam>
35          /// <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>
36          /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делагат
            ↪  <see cref="Action{T}"/>.</para></param>
37          /// <param name="maxStackSize"><para>The maximum stack size in
            ↪  bytes.</para><para>Максимальный размер стека в байтах.</para></param>
38          public static void InvokeWithModifiedMaxStackSize<T>(T param, Action<object> action, int
            ↪  maxStackSize) => StartNew(param, action, maxStackSize).Join();
39
40          /// <summary>
41          /// <para>Invokes the <see cref="Action{T}"/> with extend maximum stack size.</para>
42          /// <para>Вызывает <see cref="Action{T}"/> с расширенным максимальным размером
            ↪  стека.</para>
43          /// </summary>
44          /// <typeparam name="T"><para>The type of the <see cref="Action{T}"/>
            ↪  argument.</para><para>Тип аргумента <see cref="Action{T}"/>.</para></typeparam>
45          /// <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>
46          /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делагат
            ↪  <see cref="Action{T}"/>.</para></param>
47          public static void InvokeWithExtendedMaxStackSize<T>(T param, Action<object> action) =>
            ↪  InvokeWithModifiedMaxStackSize(param, action, DefaultExtendedMaxStackSize);
48
49          /// <summary>
50          /// <para>Invokes the <see cref="Action"/> with modified maximum stack size.</para>
51          /// <para>Вызывает <see cref="Action"/> с изменённым максимальным размером стека.</para>
52          /// </summary>
53          /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делагат
            ↪  <see cref="Action"/>.</para></param>
54          /// <param name="maxStackSize"><para>The maximum stack size in
            ↪  bytes.</para><para>Максимальный размер стека в байтах.</para></param>
55          public static void InvokeWithModifiedMaxStackSize(Action action, int maxStackSize) =>
            ↪  StartNew(action, maxStackSize).Join();
56
57          /// <summary>
58          /// <para>Invokes the <see cref="Action"/> with extend maximum stack size.</para>
59          /// <para>Вызывает <see cref="Action"/> с расширенным максимальным размером стека.</para>
60          /// </summary>
61          /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делагат
            ↪  <see cref="Action"/>.</para></param>
62          public static void InvokeWithExtendedMaxStackSize(Action action) =>
            ↪  InvokeWithModifiedMaxStackSize(action, DefaultExtendedMaxStackSize);
63
64          /// <summary>
65          /// <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>
66          /// <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>
67          /// </summary>
68          /// <typeparam name="T"><para>The type of the <see cref="Action{T}"/>
            ↪  argument.</para><para>Тип аргумента <see cref="Action{T}"/>.</para></typeparam>
69          /// <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>
70          /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делагат
            ↪  <see cref="Action{T}"/>.</para></param>
71          /// <param name="maxStackSize"><para>The maximum stack size in
            ↪  bytes.</para><para>Максимальный размер стека в байтах.</para></param>
72          /// <returns><para>A new started <see cref="Thread"/> instance.</para><para>Новый
            ↪  запущенный экземпляр <see cref="Thread"/>.</para></returns>
73          public static Thread StartNew<T>(T param, Action<object> action, int maxStackSize)
74          {
```

```
75              var thread = new Thread(new ParameterizedThreadStart(action), maxStackSize);
76              thread.Start(param);
77              return thread;
78          }
79
80          /// <summary>
81          /// <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>
82          /// <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>
83          /// </summary>
84          /// <typeparam name="T"><para>The type of the <see cref="Action{T}"/>
                ↪  argument.</para><para>Тип аргумента <see cref="Action{T}"/>.</para></typeparam>
85          /// <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>
86          /// <param name="action"><para>The <see cref="Action{T}"/> delegate.</para><para>Делагат
                ↪  <see cref="Action{T}"/>.</para></param>
87          /// <returns><para>A new started <see cref="Thread"/> instance.</para><para>Новый
                ↪  запущенный экземпляр <see cref="Thread"/>.</para></returns>
88          public static Thread StartNew<T>(T param, Action<object> action) => StartNew(param,
                ↪  action, DefaultMaxStackSize);
89
90          /// <summary>
91          /// <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>
92          /// <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>
93          /// </summary>
94          /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делагат
                ↪  <see cref="Action"/>.</para></param>
95          /// <param name="maxStackSize"><para>The maximum stack size in
                ↪  bytes.</para><para>Максимальный размер стека в байтах.</para></param>
96          /// <returns><para>A new started <see cref="Thread"/> instance.</para><para>Новый
                ↪  запущенный экземпляр <see cref="Thread"/>.</para></returns>
97          public static Thread StartNew(Action action, int maxStackSize)
98          {
99              var thread = new Thread(new ThreadStart(action), maxStackSize);
100             thread.Start();
101             return thread;
102         }
103
104         /// <summary>
105         /// <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>
106         /// <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>
107         /// </summary>
108         /// <param name="action"><para>The <see cref="Action"/> delegate.</para><para>Делагат
                ↪  <see cref="Action"/>.</para></param>
109         /// <returns><para>A new started <see cref="Thread"/> instance.</para><para>Новый
                ↪  запущенный экземпляр <see cref="Thread"/>.</para></returns>
110         public static Thread StartNew(Action action) => StartNew(action, DefaultMaxStackSize);
111
112         /// <summary>
113         /// Suspends the current thread for the <see cref="DefaultSleepInterval"/>.
114         /// </summary>
115         public static void Sleep() => Thread.Sleep(DefaultSleepInterval);
116     }
117 }
```

Index