

# LinksPlatform's Platform.IO Class Library

## 1.1 ./csharp/Platform.IO/ConsoleCancellation.cs

```
1  using System;
2  using System.Runtime.CompilerServices;
3  using System.Threading;
4  using Platform.Disposables;
5  using Platform.Threading;
6
7  namespace Platform.IO
8  {
9      /// <summary>
10     /// <para>Represents the class that simplifies the console applications implementation that
11     ↪ can be terminated manually during execution.</para>
12     /// <para>Представляет класс, упрощающий реализацию консольных приложений, выполнение
13     ↪ которых может быть прекращено в процессе выполнения вручную.</para>
14     /// </summary>
15     public class ConsoleCancellation : DisposableBase
16     {
17         /// <summary>
18         /// <para>Gets the <see cref="CancellationTokenSource"/> class instance.</para>
19         /// <para>Возвращает экземпляр класса <see cref="CancellationTokenSource"/>.</para>
20         /// </summary>
21         public CancellationTokenSource Source
22         {
23             [MethodImpl(MethodImplOptions.AggressiveInlining)]
24             get;
25         }
26
27         /// <summary>
28         /// <para>Gets the <see cref="CancellationToken"/> class instance.</para>
29         /// <para>Возвращает экземпляр класса <see cref="CancellationToken"/>.</para>
30         /// </summary>
31         public CancellationToken Token
32         {
33             [MethodImpl(MethodImplOptions.AggressiveInlining)]
34             get;
35         }
36
37         /// <summary>
38         /// <para>Gets a <see cref="Boolean"/> value that determines whether cancellation was
39         ↪ requested for the <see cref="CancellationTokenSource"/>.</para>
40         /// <para>Возвращает значение типа <see cref="Boolean"/>, определяющее, запрошена ли
41         ↪ отмена для <see cref="CancellationTokenSource"/>.</para>
42         /// </summary>
43         public bool IsRequested
44         {
45             [MethodImpl(MethodImplOptions.AggressiveInlining)]
46             get => Source.IsCancellationRequested;
47         }
48
49         /// <summary>
50         /// <para>Gets a <see cref="Boolean"/> value that determines whether cancellation was
51         ↪ not requested for the <see cref="CancellationTokenSource"/>.</para>
52         /// <para>Возвращает значение типа <see cref="Boolean"/>, определяющее, не запрошена ли
53         ↪ отмена для <see cref="CancellationTokenSource"/>.</para>
54         /// </summary>
55         public bool NotRequested
56         {
57             [MethodImpl(MethodImplOptions.AggressiveInlining)]
58             get => !Source.IsCancellationRequested;
59         }
60
61         /// <summary>
62         /// <para>Initializes a <see cref="ConsoleCancellation"/> class instance, using a <see
63         ↪ cref="CancellationTokenSource"/> and its token. The <see
64         ↪ cref="ConsoleCancellation"/> subscribes to the <see cref="Console.CancelKeyPress"/>
65         ↪ event on initialization.</para>
66         /// <para>Инициализирует экземпляр класса <see cref="ConsoleCancellation"/>, используя
67         ↪ <see cref="CancellationTokenSource"/> и его токен. <see cref="ConsoleCancellation"/>
68         ↪ подписывается на событие <see cref="Console.CancelKeyPress"/> при
69         ↪ инициализации.</para>
70         /// </summary>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         public ConsoleCancellation()
73         {
74             Source = new CancellationTokenSource();
75             Token = Source.Token;
76             Console.CancelKeyPress += OnCancelKeyPress;
77         }
78     }
79 }
```

```

66
67 /// <summary>
68 /// <para>Forces cancellation request.</para>
69 /// <para>Принудительно запрашивает отмену.</para>
70 /// </summary>
71 [MethodImpl(MethodImplOptions.AggressiveInlining)]
72 public void ForceCancellation() => Source.Cancel();
73
74 /// <summary>
75 /// <para>Suspends the current thread until a cancellation is requested.</para>
76 /// <para>Приостанавливает текущий поток до запроса на отмену.</para>
77 /// </summary>
78 [MethodImpl(MethodImplOptions.AggressiveInlining)]
79 public void Wait()
80 {
81     while (NotRequested)
82     {
83         ThreadHelpers.Sleep();
84     }
85 }
86
87 /// <summary>
88 /// <para>Unsubscribes from the <see cref="Console.CancelKeyPress"/> event and attempts
89   → to dispose the <see cref="CancellationTokenSource"/>.</para>
90 /// <para>Отписывается от события <see cref="Console.CancelKeyPress"/> и пытается
91   → высвободить ресурсы, используемые <see cref="CancellationTokenSource"/>.</para>
92 /// </summary>
93 /// <param name="manual">
94   → <para>A <see cref="Boolean"/> value that determines whether the disposal was
95   → triggered manually (by the developer's code) or was executed automatically without
96   → an explicit indication from a developer.</para>
97   → <para>Значение типа <see cref="Boolean"/>, определяющие было ли высвобождение
98   → вызвано вручную (кодом разработчика) или же выполнилось автоматически без явного
99   → указания со стороны разработчика.</para>
100 /// </param>
101 /// <param name="wasDisposed">
102   → <para>A <see cref="Boolean"/> value that determines whether the <see
103   → cref="ConsoleCancellation"/> was released before a call to this method.</para>
104   → <para>Значение типа <see cref="Boolean"/>, определяющие были ли освобождены ресурсы,
105   → используемые <see cref="ConsoleCancellation"/> до вызова данного метода.</para>
106 /// </param>
107 [MethodImpl(MethodImplOptions.AggressiveInlining)]
108 protected override void Dispose(bool manual, bool wasDisposed)
109 {
110     if (!wasDisposed)
111     {
112         Console.CancelKeyPress -= OnCancelKeyPress;
113         Source.DisposeIfPossible();
114     }
115 }
116
117 /// <summary>
118 /// <para>
119   → <para>Ons the cancel key press using the specified sender.
120   → </para>
121   → <para></para>
122   → </summary>
123 /// <param name="sender">
124   → <para>The sender.</para>
125   → <para></para>
126   → </param>
127 /// <param name="e">
128   → <para>The .</para>
129   → <para></para>
130   → </param>
131 [MethodImpl(MethodImplOptions.AggressiveInlining)]
132 private void OnCancelKeyPress(object sender, ConsoleCancelEventArgs e)
133 {
134     e.Cancel = true;
135     if (NotRequested)
136     {
137         Source.Cancel();
138     }
139 }

```

## 1.2 ./csharp/Platform.IO/ConsoleHelpers.cs

```
1 using System;
2 using System.Diagnostics;
3 using System.Runtime.CompilerServices;
4 using Platform.Collections;
5 using Platform.Collections.Arrays;
6
7 namespace Platform.IO
8 {
9     /// <summary>
10    /// <para>Represents the set of helper methods to work with the console.</para>
11    /// <para>Представляет набор вспомогательных методов для работы с консолью.</para>
12    /// </summary>
13    public static class ConsoleHelpers
14    {
15        /// <summary>
16        /// <para>Requests and expects a user to press any key in the console.</para>
17        /// <para>Запрашивает и ожидает нажатие любой клавиши пользователем в консоли.</para>
18        /// </summary>
19        [MethodImpl(MethodImplOptions.AggressiveInlining)]
20        public static void PressAnyKeyToContinue()
21        {
22            Console.WriteLine("Press any key to continue.");
23            Console.ReadKey();
24        }
25
26        /// <summary>
27        /// <para>Gets an argument's value with the specified <paramref name="index"/> from the
28        → <paramref name="args"/> array and if it's absent requests a user to input it in the
29        → console.</para>
30        /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива
31        → <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у
32        → пользователя.</para>
33        /// </summary>
34        /// <param name="index">
35        /// <para>The ordinal number of the argument in the array.</para>
36        /// <para>Порядковый номер аргумента в массиве.</para>
37        /// </param>
38        /// <param name="args">
39        /// <para>The argument array passed to the application.</para>
40        /// <para>Массив аргументов переданных приложению.</para>
41        /// </param>
42        /// <returns>
43        /// <para>The value with the specified <paramref name="index"/> extracted from the
44        → <paramref name="args"/> array or entered by a user in the console.</para>
45        /// <para>Значение с указанным <paramref name="index"/>, извлечённое из массива
46        → <paramref name="args"/>, или введённое пользователем в консоли.</para>
47        /// </returns>
48        [MethodImpl(MethodImplOptions.AggressiveInlining)]
49        public static string GetOrReadArgument(int index, params string[] args) =>
50        → GetOrReadArgument(index, $"{index + 1} argument", args);
51
52        /// <summary>
53        /// <para>Gets an argument's value with the specified <paramref name="index"/> from the
54        → <paramref name="args"/> array and if it's absent requests a user to input it in the
55        → console.</para>
56        /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива
57        → <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у
58        → пользователя.</para>
59        /// </summary>
60        /// <param name="index">
61        /// <para>The ordinal number of the argument in the array.</para>
62        /// <para>Порядковый номер аргумента в массиве.</para>
63        /// </param>
64        /// <param name="readMessage">
65        /// <para>The message's text to a user describing which argument is being entered at the
66        → moment. If the <paramref name="args"/> array doesn't contain the element with the
67        → specified <paramref name="index"/>, then this message is used.</para>
68        /// <para>Текст сообщения пользователю описывающее какой аргумент вводится в данный
69        → момент. Это сообщение используется только если массив <paramref name="args"/> не
70        → содержит аргумента с указанным <paramref name="index"/>.</para>
71        /// </param>
72        /// <param name="args">
73        /// <para>The argument array passed to the application.</para>
74        /// <para>Массив аргументов переданных приложению.</para>
75        /// </param>
76        /// <returns>
```

```

62    /// <para>The value with the specified <paramref name="index"/> extracted from the
63    → <paramref name="args"/> array or entered by a user in the console.</para>
64    /// <para>Значение с указанным <paramref name="index"/>, извлечённое из массива
65    → <paramref name="args"/>, или введённое пользователем в консоли.</para>
66    /// </returns>
67    [MethodImpl(MethodImplOptions.AggressiveInlining)]
68    public static string GetOrReadArgument(int index, string readMessage, params string[]
69    → args)
70    {
71        if (!args.TryGetElement(index, out string result))
72        {
73            Console.Write($"{readMessage}: ");
74            result = Console.ReadLine();
75        }
76        if (string.IsNullOrEmpty(result))
77        {
78            return "";
79        }
80        else
81        {
82            return result.Trim().TrimSingle(' ').Trim();
83        }
84    }
85
86    /// <summary>
87    /// <para>Outputs the <paramref name="string"/> to the console.</para>
88    /// <para>Выводит <paramref name="string"/> в консоль.</para>
89    /// </summary>
90    /// <param name="string">
91    /// <para>The string to output to the console.</para>
92    /// <para>Строка выводимая в консоль.</para>
93    /// </param>
94    /// <remarks>
95    /// <para>The method is only executed if the application was compiled with the DEBUG
96    → directive.</para>
97    /// <para>Метод выполняется только если приложение было скомпилировано с директивой
98    → DEBUG.</para>
99    /// </remarks>
100    [Conditional("DEBUG")]
101    public static void Debug(string @string) => Console.WriteLine(@string);
102
103    /// <summary>
104    /// <para>Writes text representations of the specified <paramref name="args"/> array
105    → objects to the standard output stream using the specified <paramref name="format"/>,
106    → followed by the current line terminator.</para>
107    /// <para>Записывает текстовые представления объектов заданного массива <paramref
108    → name="args"/>, в стандартный выходной поток с использованием заданного <paramref
109    → name="format"/>, за которым следует текущий признак конца строки.</para>
110    /// </summary>
111    /// <param name="format">
112    /// <para>The composite format string.</para>
113    /// <para>Строка составного формата.</para>
114    /// </param>
115    /// <param name="args">
116    /// <para>The object array to write to the standard output stream using <paramref
117    → name="format"/>.</para>
118    /// <para>Массив объектов для записи в стандартный выходной поток с использованием
119    → <paramref name="format"/>.</para>
120    /// </param>
121    /// <remarks>
122    /// <para>The method is only executed if the application was compiled with the DEBUG
123    → directive.</para>
124    /// <para>Метод выполняется только если приложение было скомпилировано с директивой
125    → DEBUG.</para>
126    /// </remarks>
127    [Conditional("DEBUG")]
128    public static void Debug(string format, params object[] args) =>
129    → Console.WriteLine(format, args);
130
131 }
132 }

```

### 1.3 ./csharp/Platform.IO/FileHelpers.cs

```

1  using System;
2  using System.IO;
3  using System.Runtime.CompilerServices;
4  using Platform.Unsafe;
5

```

```

6 namespace Platform.IO
7 {
8     /// <summary>
9     /// <para>Represents the set of helper methods to work with files.</para>
10    /// <para>Представляет набор вспомогательных методов для работы с файлами.</para>
11    /// </summary>
12    public static class FileHelpers
13    {
14        /// <summary>
15        /// <para>Reads all the text and returns character array from a file at the <paramref
16        → name="path"/>.</para>
17        /// <para>Читает весь текст и возвращает массив символов из файла находящегося в
18        → <paramref name="path"/>.</para>
19        /// </summary>
20        /// <param name="path">
21        /// <para>The path to a file, from which to read the character array.</para>
22        /// <para>Путь к файлу, из которого нужно прочитат массив символов.</para>
23        /// </param>
24        /// <returns>
25        /// <para>The character array from a file at the <paramref name="path"/>.</para>
26        /// <para>Массив символов из файла находящегося в <paramref name="path"/>.</para>
27        /// </returns>
28        [MethodImpl(MethodImplOptions.AggressiveInlining)]
29        public static char[] ReadAllChars(string path) => File.ReadAllText(path).ToCharArray();
30
31        /// <summary>
32        /// <para>Reads and returns all <typeparamref name="T"/> structure values from a file at
33        → the <paramref name="path"/>.</para>
34        /// <para>Считывает и возвращает все значения структур типа <typeparamref name="T"/> из
35        → файла находящегося в <paramref name="path"/>.</para>
36        /// </summary>
37        /// <typeparam name="T">
38        /// <para>The structure type.</para>
39        /// <para>Тип структуры.</para>
40        /// </typeparam>
41        /// <param name="path">
42        /// <para>The path to a file, from which to read <typeparamref name="T"/> structure
43        → values array.</para>
44        /// <para>Путь к файлу, из которого нужно прочитат массив значений структур типа
45        → <typeparamref name="T"/>.</para>
46        /// </param>
47        /// <returns>
48        /// <para>The <typeparamref name="T"/> structure values array.</para>
49        /// <para>Массив значений структур типа <typeparamref name="T"/>.</para>
50        /// </returns>
51        [MethodImpl(MethodImplOptions.AggressiveInlining)]
52        public static T[] ReadAll<T>(string path)
53        where T : struct
54        {
55            using var reader = File.OpenRead(path);
56            return reader.ReadAll<T>();
57        }
58
59        /// <summary>
60        /// <para>Reads and returns the first <typeparamref name="T"/> structure value from a
61        → file at the <paramref name="path"/>.</para>
62        /// <para>Считывает и возвращает первое значение структуры типа <typeparamref name="T"/>
63        → из файла находящегося в <paramref name="path"/>.</para>
64        /// </summary>
65        /// <typeparam name="T">
66        /// <para>The structure type.</para>
67        /// <para>Тип структуры.</para>
68        /// </typeparam>
69        /// <param name="path">
70        /// <para>The path to a file, from which to read the first <typeparamref name="T"/>
71        → structure value.</para>
72        /// <para>Путь к файлу, из которого нужно прочитат значение первой структуры типа
73        → <typeparamref name="T"/>.</para>
74        /// </param>
75        /// <returns>
76        /// <para>The <typeparamref name="T"/> structure value if read from a file at the
77        → <paramref name="path"/> is successful; otherwise the default <typeparamref
78        → name="T"/> structure value.</para>
79        /// <para>Значение структуры типа <typeparamref name="T"/> если чтение из файла
80        → находящегося в <paramref name="path"/> прошло успешно, иначе значение структуры типа
81        → <typeparamref name="T"/> по умолчанию.</para>
82        /// </returns>

```

```

69 [MethodImpl(MethodImplOptions.AggressiveInlining)]
70 public static T ReadFirstOrDefault<T>(string path)
71     where T : struct
72 {
73     using var fileStream = GetValidFileStreamOrDefault<T>(path);
74     return fileStream?.ReadOrDefault<T>() ?? default;
75 }
76
77 /// <summary>
78 /// <para>Returns the <see cref="FileStream"/> opened for reading from a file at the
79     ↳ <paramref name="path"/> if the file exists, not empty and its size is a multiple of
80     ↳ the <typeparamref name="TStruct"/> structure size; otherwise <see
81     ↳ langword="null"/>.</para>
82 /// <para>Возвращает <see cref="FileStream"/>, открытый для чтения из файла находящегося
83     ↳ в <paramref name="path"/>, если файл существует, не пуст и его размер кратен размеру
84     ↳ структуры типа <typeparamref name="TStruct"/>, а иначе <see langword="null"/>.</para>
85 /// </summary>
86 /// <typeparam name="TStruct">
87 /// <para>The structure type.</para>
88 /// <para>Тип структуры.</para>
89 /// </typeparam>
90 /// <param name="path">
91 /// <para>The path to a file to validate.</para>
92 /// <para>Путь к проверяемому файлу.</para>
93 /// </param>
94 /// <returns>
95 /// <para>A <see cref="FileStream"/> opened for reading in the case of successful check;
96     ↳ otherwise <see langword="null"/>.</para>
97 /// <para><see cref="FileStream"/>, открытый для чтения в случае успешной проверки, а
98     ↳ иначе <see langword="null"/>.</para>
99 /// </returns>
100 /// <exception cref="InvalidOperationException">
101 /// <para>The size of a file at the <paramref name="path"/> is not a multiple of the
102     ↳ required <typeparamref name="TStruct"/> structure size.</para>
103 /// <para>Размер файла находящегося в <paramref name="path"/> не кратен требуемому
104     ↳ размеру структуры типа <typeparamref name="TStruct"/>.</para>
105 /// </exception>
106 [MethodImpl(MethodImplOptions.AggressiveInlining)]
107 private static FileStream GetValidFileStreamOrDefault<TStruct>(string path) where
108     ↳ TStruct : struct => GetValidFileStreamOrDefault(path, Structure<TStruct>.Size);
109
110 /// <summary>
111 /// <para>Returns the <see cref="FileStream"/> opened for reading from a file at the
112     ↳ <paramref name="path"/> if the file exists, not empty and its size is a multiple of
113     ↳ the required <paramref name="elementSize"/>; otherwise <see langword="null"/>.</para>
114 /// <para>Возвращает <see cref="FileStream"/>, открытый для чтения из файла находящегося
115     ↳ в <paramref name="path"/>, если файл существует, не пуст и его размер кратен
116     ↳ <paramref name="elementSize"/>, а иначе <see langword="null"/>.</para>
117 /// </summary>
118 /// <param name="path">
119 /// <para>The path to a file to validate.</para>
120 /// <para>Путь к проверяемому файлу.</para>
121 /// </param>
122 /// <param name="elementSize">
123 /// <para>Required size of elements located in a file at the <paramref
124     ↳ name="path"/>.</para>
125 /// <para>Требуемый размер элементов, находящихся в файле находящегося в <paramref
126     ↳ name="path"/>.</para>
127 /// </param>
128 /// <returns>
129 /// <para>A <see cref="FileStream"/> opened for reading in the case of successful check;
130     ↳ otherwise <see langword="null"/>.</para>
131 /// <para><see cref="FileStream"/>, открытый для чтения в случае успешной проверки, а
132     ↳ иначе <see langword="null"/>.</para>
133 /// </returns>
134 /// <exception cref="InvalidOperationException">
135 /// <para>The size of a file at the <paramref name="path"/> is not a multiple of the
136     ↳ required <paramref name="elementSize"/>.</para>
137 /// <para>Размер файла находящегося в <paramref name="path"/> не кратен требуемому
138     ↳ <paramref name="elementSize"/>.</para>
139 /// </exception>
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 private static FileStream GetValidFileStreamOrDefault(string path, int elementSize)
142 {
143     if (!File.Exists(path))
144     {

```

```

125         return null;
126     }
127     var fileSize = GetSize(path);
128     if (fileSize % elementSize != 0)
129     {
130         throw new InvalidOperationException($"File is not aligned to elements with size
131         ↳ {elementSize}.");
132     }
133     return fileSize > 0 ? File.OpenRead(path) : null;
134 }
135
136 /// <summary>
137 /// <para>Reads and returns the last <typeparamref name="T"/> structure value from a
138 ↳ file at the <paramref name="path"/>.</para>
139 /// <para>Считывает и возвращает последнее значение структуры типа <typeparamref
140 ↳ name="T"/> из файла находящегося в <paramref name="path"/>.</para>
141 /// </summary>
142 /// <typeparam name="T">
143 /// <para>The structure type.</para>
144 /// <para>Тип структуры.</para>
145 /// </typeparam>
146 /// <param name="path">
147 /// <para>The path to a file, from which to read the last <typeparamref name="T"/>
148 ↳ structure value.</para>
149 /// <para>Путь к файлу, из которого нужно прочесть значение последней структуры типа
150 ↳ <typeparamref name="T"/>.</para>
151 /// </param>
152 /// <returns>
153 /// <para>The <typeparamref name="T"/> structure value if read from a file at the
154 ↳ <paramref name="path"/> is successful; otherwise the default <typeparamref
155 ↳ name="T"/> structure value.</para>
156 /// <para>Значение структуры типа <typeparamref name="T"/> из файла находящегося в
157 ↳ <paramref name="path"/> в случае успешного чтения, иначе значение по умолчанию
158 ↳ структуры типа <typeparamref name="T"/>.</para>
159 /// </returns>
160 [MethodImpl(MethodImplOptions.AggressiveInlining)]
161 public static T ReadLastOrDefault<T>(string path)
162     where T : struct
163 {
164     var elementSize = Structure<T>.Size;
165     using var reader = GetValidFileStreamOrDefault(path, elementSize);
166     if (reader == null)
167     {
168         return default;
169     }
170     var totalElements = reader.Length / elementSize;
171     reader.Position = (totalElements - 1) * elementSize; // Set to last element
172     return reader.ReadOrDefault<T>();
173 }
174
175 /// <summary>
176 /// <para>Writes <typeparamref name="T"/> structure value at the beginning of a file at
177 ↳ the <paramref name="path"/>.</para>
178 /// <para>Записывает значение структуры типа <typeparamref name="T"/> в начало файла
179 ↳ находящегося в <paramref name="path"/>.</para>
180 /// </summary>
181 /// <typeparam name="T">
182 /// <para>The structure type.</para>
183 /// <para>Тип структуры.</para>
184 /// </typeparam>
185 /// <param name="path">
186 /// <para>The path to a file to be changed or created.</para>
187 /// <para>Путь к файлу, который будет изменён или создан.</para>
188 /// </param>
189 /// <param name="value">
190 /// <para>The <typeparamref name="T"/> structure value to be written at the beginning of
191 ↳ a file at the <paramref name="path"/>.</para>
192 /// <para>Значение структуры типа <typeparamref name="T"/>, записываемое в начало файла
193 ↳ находящегося в <paramref name="path"/>.</para>
194 /// </param>
195 [MethodImpl(MethodImplOptions.AggressiveInlining)]
196 public static void WriteFirst<T>(string path, T value)
197     where T : struct
198 {
199     using var writer = File.OpenWrite(path);
200     writer.Position = 0;
201     writer.Write(value);
202 }

```

```

189     }
190
191     /// <summary>
192     /// <para>Opens or creates a file at the <paramref name="path"/> and returns its <see
193     → cref="FileStream"/> with append mode and write access.</para>
194     /// <para>Открывает или создаёт файл находящийся в <paramref name="path"/> и возвращает
195     → его <see cref="FileStream"/> с режимом дополнения и доступом на запись.</para>
196     /// </summary>
197     /// <param name="path">
198     /// <para>The path to a file to open or create.</para>
199     /// <para>Путь к файлу, который нужно открыть или создать.</para>
200     /// </param>
201     /// <returns>
202     /// <para>The <see cref="FileStream"/> with append mode and write access.</para>
203     /// <para><see cref="FileStream"/> с режимом дополнения и доступом на запись.</para>
204     /// </returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     public static FileStream Append(string path) => File.Open(path, FileMode.Append,
207     → FileAccess.Write);
208
209     /// <summary>
210     /// <para>Returns the size of a file at the <paramref name="path"/> if the file exists;
211     → otherwise 0.</para>
212     /// <para>Возвращает размер файла находящегося в <paramref name="path"/> если тот
213     → существует, иначе 0.</para>
214     /// </summary>
215     /// <param name="path">
216     /// <para>The path to a file to get size.</para>
217     /// <para>Путь к файлу, размер которого нужно получить.</para>
218     /// </param>
219     /// <returns>
220     /// <para>Size of a file at the <paramref name="path"/> if it exists; otherwise 0.</para>
221     /// <para>Размер файла если файл находящийся в <paramref name="path"/> существует, иначе
222     → 0.</para>
223     /// </returns>
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     public static long GetSize(string path) => File.Exists(path) ? new FileInfo(path).Length
226     → : 0;
227
228     /// <summary>
229     /// <para>Sets the <paramref name="size"/> for a file at the <paramref
230     → name="path"/>.</para>
231     /// <para>Устанавливает <paramref name="size"/> файлу находящемуся по пути <paramref
232     → name="path"/>.</para>
233     /// </summary>
234     /// <param name="path">
235     /// <para>The path to a file to be resized.</para>
236     /// <para>Путь к файлу, размер которого нужно изменить.</para>
237     /// </param>
238     /// <param name="size">
239     /// <para>The size to assign to a file at the <paramref name="path"/>.</para>
240     /// <para>Размер который будет присвоен файлу находящемуся по пути <paramref
241     → name="path"/>.</para>
242     /// </param>
243     [MethodImpl(MethodImplOptions.AggressiveInlining)]
244     public static void SetSize(string path, long size)
245     {
246         using var fileStream = File.Open(path, FileMode.OpenOrCreate);
247         if (fileStream.Length != size)
248         {
249             fileStream.SetLength(size);
250         }
251     }
252
253     /// <summary>
254     /// <para>Removes all files from the directory at the path <paramref
255     → name="directory"/>.</para>
256     /// <para>Удаляет все файлы из директории находящейся по пути <paramref
257     → name="directory"/>.</para>
258     /// </summary>
259     /// <param name="directory">
260     /// <para>The path to the directory to be cleaned.</para>
261     /// <para>Путь к директории для очистки.</para>
262     /// </param>
263     [MethodImpl(MethodImplOptions.AggressiveInlining)]
264     public static void DeleteAll(string directory) => DeleteAll(directory, "*");
265
266

```



```

254 /// <summary>
255 /// <para>Removes files from the directory at the path <paramref name="directory"/>
    ↳ according to the <paramref name="searchPattern"/>.</para>
256 /// <para>Удаляет файлы из директории находящейся по пути <paramref name="directory"/> в
    ↳ соответствии с <paramref name="searchPattern"/>.</para>
257 /// </summary>
258 /// <param name="directory">
259 /// <para>The path to the directory to be cleaned.</para>
260 /// <para>Путь к директории для очистки.</para>
261 /// </param>
262 /// <param name="searchPattern">
263 /// <para>The search pattern for files to be deleted in the directory at the path
    ↳ <paramref name="directory"/>.</para>
264 /// <para>Шаблон поиска для удаляемых файлов в директории находящейся по пути <paramref
    ↳ name="directory"/>.</para>
265 /// </param>
266 [MethodImpl(MethodImplOptions.AggressiveInlining)]
267 public static void DeleteAll(string directory, string searchPattern) =>
    ↳ DeleteAll(directory, searchPattern, SearchOption.TopDirectoryOnly);
268
269 /// <summary>
270 /// <para>Removes files from the directory at the path <paramref name="directory"/>
    ↳ according to the <paramref name="searchPattern"/> and the <paramref
    ↳ name="searchOption"/>.</para>
271 /// <para>Удаляет файлы из директории находящейся по пути <paramref name="directory"/> в
    ↳ соответствии с <paramref name="searchPattern"/> и <paramref
    ↳ name="searchOption"/>.</para>
272 /// </summary>
273 /// <param name="directory">
274 /// <para>The path to the directory to be cleaned.</para>
275 /// <para>Путь к директории для очистки.</para>
276 /// </param>
277 /// <param name="searchPattern">
278 /// <para>The search pattern for files to be deleted in the directory at the path
    ↳ <paramref name="directory"/>.</para>
279 /// <para>Шаблон поиска для удаляемых файлов в директории находящейся по пути <paramref
    ↳ name="directory"/> .</para>
280 /// </param>
281 /// <param name="searchOption">
282 /// <para>The <see cref="SearchOption"/> value that determines whether to search only in
    ↳ the current the directory at the path <paramref name="directory"/>, or also in all
    ↳ subdirectories.</para>
283 /// <para>Значение <see cref="SearchOption"/> определяющее искать ли только в текущей
    ↳ директории находящейся по пути <paramref name="directory"/>, или также во всех
    ↳ субдиректориях.</para>
284 /// </param>
285 [MethodImpl(MethodImplOptions.AggressiveInlining)]
286 public static void DeleteAll(string directory, string searchPattern, SearchOption
    ↳ searchOption)
287 {
288     foreach (var file in Directory.EnumerateFiles(directory, searchPattern,
    ↳ searchOption))
289     {
290         File.Delete(file);
291     }
292 }
293
294 /// <summary>
295 /// <para>Truncates the file at the <paramref name="path"/>.</para>
296 /// <para>Очищает содержимое файла по пути <paramref name="path"/>.</para>
297 /// </summary>
298 /// <param name="path">
299 /// <para>A path to a file to be truncated.</para>
300 /// <para>Путь к файлу для очистки содержимого.</para>
301 /// </param>
302 [MethodImpl(MethodImplOptions.AggressiveInlining)]
303 public static void Truncate(string path) => File.Open(path, FileMode.Truncate).Dispose();
304
305 /// <summary>
306 /// <para>Appends the <paramref name="content"/> to a file at the <paramref
    ↳ name="path"/>.</para>
307 /// <para>Добавляет <paramref name="content"/> в конец файла по пути <paramref
    ↳ name="path"/>.</para>
308 /// </summary>
309 /// <param name="path">
310 /// <para>The path to a file to be appended by the <paramref name="content"/>.</para>

```

```

311     /// <para>Путь к файлу для добавления <paramref name="content"/> в конец файла.</para>
312     /// </param>
313     /// <param name="content">
314     /// <para>A content to be appended to a file at the file at the <paramref
    → name="path"/>.</para>
315     /// <para>Содержимое для добавления в конец файла по пути <paramref name="path"/>.</para>
316     /// </param>
317     [MethodImpl(MethodImplOptions.AggressiveInlining)]
318     public static void AppendLine(string path, string content)
319     {
320         using var writer = File.AppendText(path);
321         writer.WriteLine(content);
322     }
323
324     /// <summary>
325     /// <para>Performs the <paramref name="action"/> for each line of a file at the
    → <paramref name="path"/>.</para>
326     /// <para>Выполняет <paramref name="action"/> для каждой строки файла по пути <paramref
    → name="path"/>.</para>
327     /// </summary>
328     /// <param name="path">
329     /// <para>A path to a file to perform the <paramref name="action"/> for each line of the
    → file.</para>
330     /// <para>Путь к файлу для выполнения <paramref name="action"/> для каждой строки
    → файла.</para>
331     /// </param>
332     /// <param name="action">
333     /// <para>An action to be performed for each line of a file at the <paramref
    → name="path"/>.</para>
334     /// <para>Действие выполняемое для каждой строки файла по пути <paramref
    → name="path"/>.</para>
335     /// </param>
336     [MethodImpl(MethodImplOptions.AggressiveInlining)]
337     public static void EachLine(string path, Action<string> action)
338     {
339         using var reader = new StreamReader(path);
340         string line;
341         while ((line = reader.ReadLine()) != null)
342         {
343             action(line);
344         }
345     }
346 }
347 }

```

#### 1.4 ./csharp/Platform.IO/StreamExtensions.cs

```

1  using System.IO;
2  using System.Runtime.CompilerServices;
3  using Platform.Unsafe;
4
5  namespace Platform.IO
6  {
7      /// <summary>
8      /// <para>Represents the set of extension methods for <see cref="Stream"/> class
    → instances.</para>
9      /// <para>Представляет набор методов расширения для экземпляров класса <see
    → cref="Stream"/>.</para>
10     /// </summary>
11     public static class StreamExtensions
12     {
13         /// <summary>
14         /// <para>Writes a byte sequence that represents the <typeparamref name="T"/> structure
    → <paramref name="value"/> to the <paramref name="stream"/> and moves the current
    → position of the <paramref name="stream"/> by the number of written bytes.</para>
15         /// <para>Записывает последовательность байт представляющую <paramref name="value"/>
    → структуры типа <typeparamref name="T"/> в поток <paramref name="stream"/> и
    → перемещает текущую позицию в <paramref name="stream"/> вперёд на число записанных
    → байт.</para>
16         /// </summary>
17         /// <typeparam name="T">
18         /// <para>The structure type.</para>
19         /// <para>Тип структуры.</para>
20         /// </typeparam>
21         /// <param name="stream">
22         /// <para>The stream to write to.</para>
23         /// <para>Поток, в который осуществляется запись.</para>
24         /// </param>

```

```

25 /// <param name="value">
26 /// <para>The <typeparamref name="T"/> structure value to be written to the <paramref
   → name="stream"/>.</para>
27 /// <para>Значение структуры типа <typeparamref name="T"/> которое будет записано в
   → поток <paramref name="stream"/>.</para>
28 /// </param>
29 [MethodImpl(MethodImplOptions.AggressiveInlining)]
30 public static void Write<T>(this Stream stream, T value)
31     where T : struct
32 {
33     var bytes = value.ToBytes();
34     stream.Write(bytes, 0, bytes.Length);
35 }
36
37 /// <summary>
38 /// <para>Reads a byte sequence that represents the <typeparamref name="T"/> structure
   → value and moves the current position of the <paramref name="stream"/> by the number
   → of read bytes.</para>
39 /// <para>Считывает последовательность байт представляющих значение структуры типа
   → <typeparamref name="T"/> и перемещает текущую позицию в потоке <paramref
   → name="stream"/> вперёд на число прочитанных байт.</para>
40 /// </summary>
41 /// <typeparam name="T">
42 /// <para>The structure type.</para>
43 /// <para>Тип структуры.</para>
44 /// </typeparam>
45 /// <param name="stream">
46 /// <para>The stream containing the <typeparamref name="T"/> structure value.</para>
47 /// <para>Поток, содержащий значение структуры типа <typeparamref name="T"/>.</para>
48 /// </param>
49 /// <returns>
50 /// <para>The <typeparamref name="T"/> structure value, if its bytes from the <paramref
   → name="stream"/> are read; otherwise the default <typeparamref name="T"/> structure
   → value.</para>
51 /// <para>Значение структуры типа <typeparamref name="T"/>, если её байты из потока
   → <paramref name="stream"/> были прочитаны, иначе значение структуры типа
   → <typeparamref name="T"/> по умолчанию.</para>
52 /// </returns>
53 [MethodImpl(MethodImplOptions.AggressiveInlining)]
54 public static T ReadOrDefault<T>(this Stream stream)
55     where T : struct
56 {
57     var size = Structure<T>.Size;
58     var buffer = new byte[size];
59     return stream.Read(buffer, 0, size) == size ? buffer.ToStructure<T>() : default;
60 }
61
62 /// <summary>
63 /// <para>Reads and returns all <typeparamref name="T"/> structure values array from the
   → <paramref name="stream"/>.</para>
64 /// <para>Прочитывает и возвращает массив всех значений структур типа <typeparamref
   → name="T"/> из потока <paramref name="stream"/>.</para>
65 /// </summary>
66 /// <typeparam name="T">
67 /// <para>The structure type.</para>
68 /// <para>Тип структуры.</para>
69 /// </typeparam>
70 /// <param name="stream">
71 /// <para>The stream containing the <typeparamref name="T"/> structure values.</para>
72 /// <para>Поток, содержащий значения структур типа <typeparamref name="T"/>.</para>
73 /// </param>
74 /// <returns>
75 /// <para>The <typeparamref name="T"/> structure values array read from the <paramref
   → name="stream"/>.</para>
76 /// <para>Массив с значениями структур типа <typeparamref name="T"/>, прочитанными из
   → потока <paramref name="stream"/>.</para>
77 /// </returns>
78 [MethodImpl(MethodImplOptions.AggressiveInlining)]
79 public static T[] ReadAll<T>(this Stream stream)
80     where T : struct
81 {
82     var size = Structure<T>.Size;
83     var buffer = new byte[size];
84     var elementsLength = stream.Length / size;
85     var elements = new T[elementsLength];
86     for (var i = 0; i < elementsLength; i++)
87     {

```

```

88         stream.Read(buffer, 0, size);
89         elements[i] = buffer.ToStructure<T>();
90     }
91     return elements;
92 }
93 }
94 }

```

## 1.5 ./csharp/Platform.IO/TemporaryFile.cs

```

1  using Platform.Disposables;
2  using System;
3  using System.IO;
4  using System.Runtime.CompilerServices;
5
6  namespace Platform.IO
7  {
8      /// <summary>
9      /// <para>Represents a self-deleting temporary file.</para>
10     /// <para>Представляет самоудаляющийся временный файл.</para>
11     /// </summary>
12     public class TemporaryFile : DisposableBase
13     {
14         /// <summary>
15         /// <para>Gets a temporary file path.</para>
16         /// <para>Возвращает путь к временному файлу.</para>
17         /// </summary>
18         public readonly string Filename;
19
20         /// <summary>
21         /// <para>Converts the <see cref="TemporaryFile"/> instance to <see cref="string"/>
22         → using the <see cref="Filename"/> field value.</para>
23         /// <para>Преобразует экземпляр <see cref="TemporaryFile"/> в <see cref="string"/>
24         → используя поле <see cref="Filename"/>.</para>
25         /// </summary>
26         /// <param name="file">
27         /// <para>A <see cref="TemporaryFile"/> instance.</para>
28         /// <para>Экземпляр <see cref="TemporaryFile"/>.</para>
29         /// </param>
30         /// <returns>
31         /// <para>Path to the temporary file.</para>
32         /// <para>Путь к временному файлу.</para>
33         /// </returns>
34         public static implicit operator string(TemporaryFile file) => file.Filename;
35
36         /// <summary>
37         /// <para>Initializes a <see cref="TemporaryFile"/> instance.</para>
38         /// <para>Инициализирует экземпляр класса <see cref="TemporaryFile"/>.</para>
39         /// </summary>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public TemporaryFile() => Filename = TemporaryFiles.UseNew();
42
43         /// <summary>
44         /// <para>Deletes the temporary file.</para>
45         /// <para>Удаляет временный файл.</para>
46         /// </summary>
47         /// <param name="manual">
48         /// <para>A <see cref="Boolean"/> value that determines whether the disposal was
49         → triggered manually (by the developer's code) or was executed automatically without
50         → an explicit indication from a developer.</para>
51         /// <para>Значение типа <see cref="Boolean"/>, определяющие было ли высвобождение
52         → вызвано вручную (кодом разработчика) или же выполнилось автоматически без явного
53         → указания со стороны разработчика.</para>
54         /// </param>
55         /// <param name="wasDisposed">
56         /// <para>A <see cref="Boolean"/> value that determines whether the <see
57         → cref="TemporaryFile"/> was released before a call to this method.</para>
58         /// <para>Значение типа <see cref="Boolean"/>, определяющие были ли освобождены ресурсы,
59         → используемые <see cref="TemporaryFile"/> до вызова данного метода.</para>
60         /// </param>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         protected override void Dispose(bool manual, bool wasDisposed)
63         {
64             if (!wasDisposed)
65             {
66                 File.Delete(Filename);
67             }
68         }
69     }
70 }

```

```
62 }
```

## 1.6 ./csharp/Platform.IO/TemporaryFiles.cs

```
1 using System.IO;
2 using System.Reflection;
3 using System.Runtime.CompilerServices;
4
5 namespace Platform.IO
6 {
7     /// <summary>
8     /// <para>Represents the set of helper methods to work with temporary files.</para>
9     /// <para>Представляет набор вспомогательных методов для работы с временными файлами.</para>
10    /// </summary>
11    public class TemporaryFiles
12    {
13        /// <summary>
14        /// <para>
15        /// The user files list file name prefix.
16        /// </para>
17        /// <para></para>
18        /// </summary>
19        private const string UserFilesListFileNamePrefix = ".used-temporary-files.txt";
20        /// <summary>
21        /// <para>
22        /// The used files list lock.
23        /// </para>
24        /// <para></para>
25        /// </summary>
26        private static readonly object UsedFilesListLock = new();
27        /// <summary>
28        /// <para>
29        /// The user files list file name prefix.
30        /// </para>
31        /// <para></para>
32        /// </summary>
33        private static readonly string UsedFilesListFilename =
34            ↪ Assembly.GetExecutingAssembly().Location + UserFilesListFileNamePrefix;
35        /// <summary>
36        /// <para>
37        /// Adds the to used files list using the specified filename.
38        /// </para>
39        /// <para></para>
40        /// </summary>
41        /// <param name="filename">
42        /// <para>The filename.</para>
43        /// <para></para>
44        /// </param>
45        [MethodImpl(MethodImplOptions.AggressiveInlining)]
46        private static void AddToUsedFilesList(string filename)
47        {
48            lock (UsedFilesListLock)
49            {
50                FileHelpers.AppendLine(UsedFilesListFilename, filename);
51            }
52        }
53
54        /// <summary>
55        /// <para>Gets a temporary file and adds it to the used files list.</para>
56        /// <para>Получает временный файл и добавляет его в список использованных файлов.</para>
57        /// </summary>
58        /// <returns>
59        /// <para>The temporary file path.</para>
60        /// <para>Путь временного файла.</para>
61        /// </returns>
62        [MethodImpl(MethodImplOptions.AggressiveInlining)]
63        public static string UseNew()
64        {
65            var filename = Path.GetTempFileName();
66            AddToUsedFilesList(filename);
67            return filename;
68        }
69
70        /// <summary>
71        /// <para>Deletes all previously used temporary files and clears the files list.</para>
72        /// <para>Удаляет все ранее использованные временные файлы и очищает список
73        ↪ файлов.</para>
74        /// </summary>
```

```

74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public static void DeleteAllPreviouslyUsed()
76     {
77         lock (UsedFilesListLock)
78         {
79             var listFilename = UsedFilesListFilename;
80             if (File.Exists(listFilename))
81             {
82                 FileHelpers.EachLine(listFilename, File.Delete);
83                 FileHelpers.Truncate(listFilename);
84             }
85         }
86     }
87 }
88 }

```

## 1.7 ./csharp/Platform.IO.Tests/FileHelpersTests.cs

```

1  using System.IO;
2  using Xunit;
3
4  namespace Platform.IO.Tests
5  {
6      /// <summary>
7      /// <para>
8      /// Represents the file helpers tests.
9      /// </para>
10     /// <para></para>
11     /// </summary>
12     public class FileHelpersTests
13     {
14         /// <summary>
15         /// <para>
16         /// Tests that write read test.
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         [Fact]
21         public void WriteReadTest()
22         {
23             var temporaryFile = Path.GetTempFileName();
24             var originalValue = 42UL;
25             FileHelpers.WriteFirst(temporaryFile, originalValue);
26             var readValue = FileHelpers.ReadFirstOrDefault<ulong>(temporaryFile);
27             Assert.Equal(readValue, originalValue);
28             File.Delete(temporaryFile);
29         }
30     }
31 }

```

## 1.8 ./csharp/Platform.IO.Tests/TemporaryFileTests.cs

```

1  using Xunit;
2  using System.IO;
3  using System.Diagnostics;
4
5  namespace Platform.IO.Tests
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the temporary file tests.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public class TemporaryFileTests
14     {
15         /// <summary>
16         /// <para>
17         /// Tests that temporary file test.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         [Fact]
22         public void TemporaryFileTest()
23         {
24             using Process process = new();
25             process.StartInfo.FileName =
26                 ↪ Path.GetFullPath(Path.Combine(Directory.GetCurrentDirectory(), "..", "..", "..",
27                 ↪ "...", "Platform.IO.Tests.TemporaryFileTest", "bin", "Debug", "net5",
28                 ↪ "Platform.IO.Tests.TemporaryFileTest"));
29             process.StartInfo.UseShellExecute = false;

```

```

27     process.StartInfo.RedirectStandardOutput = true;
28     process.Start();
29     var path = process.StandardOutput.ReadLine();
30     Assert.True(File.Exists(path));
31     process.WaitForExit();
32     Assert.False(File.Exists(path));
33 }
34
35 /// <summary>
36 /// <para>
37 /// Tests that temporary file test without console app.
38 /// </para>
39 /// <para></para>
40 /// </summary>
41 [Fact]
42 public void TemporaryFileTestWithoutConsoleApp()
43 {
44     string fileName;
45     using (TemporaryFile tempFile = new())
46     {
47         fileName = tempFile;
48         Assert.True(File.Exists(fileName));
49     }
50     Assert.False(File.Exists(fileName));
51 }
52 }
53 }

```

## Index

- ./csharp/Platform.IO.Tests/FileHelpersTests.cs, 14
- ./csharp/Platform.IO.Tests/TemporaryFileTests.cs, 14
- ./csharp/Platform.IO/ConsoleCancellation.cs, 1
- ./csharp/Platform.IO/ConsoleHelpers.cs, 2
- ./csharp/Platform.IO/FileHelpers.cs, 4
- ./csharp/Platform.IO/StreamExtensions.cs, 10
- ./csharp/Platform.IO/TemporaryFile.cs, 12
- ./csharp/Platform.IO/TemporaryFiles.cs, 13