

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 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.IO
10 {
11     /// <summary>
12     /// <para>Represents the class that simplifies the console applications implementation that
13     ///     ↪ can be terminated manually during execution.</para>
14     /// <para>Представляет класс, упрощающий реализацию консольных приложений, выполнение
15     ///     ↪ которых может быть прекращено в процессе выполнения вручную.</para>
16     /// </summary>
17     public class ConsoleCancellation : DisposableBase
18     {
19         /// <summary>
20         /// <para>Gets the <see cref="CancellationTokenSource"/> class instance.</para>
21         /// <para>Возвращает экземпляр класса <see cref="CancellationTokenSource"/>.</para>
22         /// </summary>
23         public CancellationTokenSource Source
24         {
25             [MethodImpl(MethodImplOptions.AggressiveInlining)]
26             get;
27         }
28
29         /// <summary>
30         /// <para>Gets the <see cref="CancellationToken"/> class instance.</para>
31         /// <para>Возвращает экземпляр класса <see cref="CancellationToken"/>.</para>
32         /// </summary>
33         public CancellationToken Token
34         {
35             [MethodImpl(MethodImplOptions.AggressiveInlining)]
36             get;
37         }
38
39         /// <summary>
40         /// <para>Gets a <see cref="Boolean"/> value that determines whether cancellation was
41         ///     ↪ requested for the <see cref="CancellationTokenSource"/>.</para>
42         /// <para>Возвращает значение типа <see cref="Boolean"/>, определяющее, запрошена ли
43         ///     ↪ отмена для <see cref="CancellationTokenSource"/>.</para>
44         /// </summary>
45         public bool IsRequested
46         {
47             [MethodImpl(MethodImplOptions.AggressiveInlining)]
48             get => Source.IsCancellationRequested;
49         }
50
51         /// <summary>
52         /// <para>Gets a <see cref="Boolean"/> value that determines whether cancellation was
53         ///     ↪ requested for the <see cref="CancellationTokenSource"/>.</para>
54         /// <para>Возвращает значение типа <see cref="Boolean"/>, определяющее, не запрошена ли
55         ///     ↪ отмена для <see cref="CancellationTokenSource"/>.</para>
56         /// </summary>
57         public bool NotRequested
58         {
59             [MethodImpl(MethodImplOptions.AggressiveInlining)]
60             get => !Source.IsCancellationRequested;
61         }
62
63         /// <summary>
64         /// <para>Initializes the <see cref="ConsoleCancellation"/> class instance, using an
65         ///     ↪ <see cref="CancellationTokenSource"/> and its token. The <see
66         ///     ↪ cref="ConsoleCancellation"/> subscribes to the <see cref="Console.CancelKeyPress"/>
67         ///     ↪ event on initialization.</para>
68         /// <para>Инициализирует экземпляр класса <see cref="ConsoleCancellation"/>, используя
69         ///     ↪ <see cref="CancellationTokenSource"/> и его токен. <see cref="ConsoleCancellation"/>
70         ///     ↪ подписывается на событие <see cref="Console.CancelKeyPress"/> при
71         ///     ↪ инициализации.</para>
72         /// </summary>
73         [MethodImpl(MethodImplOptions.AggressiveInlining)]
74         public ConsoleCancellation()
75         {
76             Source = new CancellationTokenSource();
77             Token = Source.Token;
78         }
79     }
80 }
```

```

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

```

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 the 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 the 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>
77     /// <para>The value with the specified <paramref name="index"/> extracted from the
78     ///   → <paramref name="args"/> array or entered by a user in the console.</para>
79     /// <para>Значение с указанным <paramref name="index"/>, извлечённое из массива
80     ///   → <paramref name="args"/>, или введённое пользователем в консоли.</para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     public static string GetOrReadArgument(int index, string readMessage, params string[]
84     → args)
85     {
86         if (!args.TryGetElement(index, out string result))
87         {
88             Console.Write($"{readMessage}: ");
89         }
90     }
91 }

```

```

71         result = Console.ReadLine();
72     }
73     if (string.IsNullOrEmpty(result))
74     {
75         return "";
76     }
77     else
78     {
79         return result.Trim().TrimSingle(' ').Trim();
80     }
81 }
82
83 /// <summary>
84 /// <para>Outputs the <paramref name="string"/> to the console.</para>
85 /// <para>Выводит <paramref name="string"/> в консоль.</para>
86 /// </summary>
87 /// <param name="string">
88 /// <para>The string to output to the console.</para>
89 /// <para>Строка выводимая в консоль.</para>
90 /// </param>
91 /// <remarks>
92 /// <para>The method is only executed if the application was compiled with the DEBUG
93   → directive.</para>
94 /// <para>Метод выполняется только в том случае, если приложение было скомпилировано с
95   → директивой DEBUG.</para>
96 /// </remarks>
97 [Conditional("DEBUG")]
98 public static void Debug(string @string) => Console.WriteLine(@string);
99
100 /// <summary>
101 /// <para>Writes text representations of the specified <paramref name="args"/> array
102   → objects, followed by the current line terminator, to the standard output stream
103   → using the specified <paramref name="format"/>.</para>
104 /// <para>Записывает текстовые представления объектов заданного массива <paramref
105   → name="args"/>, за которым следует текущий признак конца строки, в стандартный
106   → выходной поток с использованием заданного <paramref name="format"/>.</para>
107 /// </summary>
108 /// <param name="format">
109 /// <para>The composite format string.</para>
110 /// <para>Строка составного формата.</para>
111 /// </param>
112 /// <param name="args">
113 /// <para>The object array to write to the standard output stream using <paramref
114   → name="format"/>.</para>
115 /// <para>Массив объектов для записи в стандартный выходной поток с использованием
116   → <paramref name="format"/>.</para>
117 /// </param>
118 /// <remarks>
119 /// <para>The method is only executed if the application was compiled with the DEBUG
120   → directive.</para>
121 /// <para>Метод выполняется только в том случае, если приложение было скомпилировано с
122   → директивой DEBUG.</para>
123 /// </remarks>
124 [Conditional("DEBUG")]
125 public static void Debug(string format, params object[] args) =>
126   → Console.WriteLine(format, args);
127 }

```

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 the 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>

```

```

17     /// </summary>
18     /// <param name="path">
19     /// <para>The path to the file, from which to read the character array.</para>
20     /// <para>Путь к файлу, из которого нужно прочитать массив символов.</para>
21     /// </param>
22     /// <returns>
23     /// <para>The character array from the file at the <paramref name="path"/>.</para>
24     /// <para>Массив символов из файла находящегося в <paramref name="path"/>.</para>
25     /// </returns>
26     [MethodImpl(MethodImplOptions.AggressiveInlining)]
27     public static char[] ReadAllChars(string path) => File.ReadAllText(path).ToCharArray();
28
29     /// <summary>
30     /// <para>Reads and returns all <typeparamref name="T"/> structure values from the file
31     ///     ↳ at the <paramref name="path"/>.</para>
32     /// <para>Считывает и возвращает все значения структур типа <typeparamref name="T"/> из
33     ///     ↳ файла находящегося в <paramref name="path"/>.</para>
34     /// </summary>
35     /// <typeparam name="T">
36     /// <para>The structure type.</para>
37     /// <para>Тип структуры.</para>
38     /// </typeparam>
39     /// <param name="path">
40     /// <para>The path to the file, from which to read <typeparamref name="T"/> structure
41     ///     ↳ values array.</para>
42     /// <para>Путь к файлу, из которого нужно прочитать массив значений структур типа
43     ///     ↳ <typeparamref name="T"/>.</para>
44     /// </param>
45     /// <returns>
46     /// <para>The <typeparamref name="T"/> structure values array.</para>
47     /// <para>Массив значений структур типа <typeparamref name="T"/>.</para>
48     /// </returns>
49     [MethodImpl(MethodImplOptions.AggressiveInlining)]
50     public static T[] ReadAll<T>(string path)
51     where T : struct
52     {
53         using var reader = File.OpenRead(path);
54         return reader.ReadAll<T>();
55     }
56
57     /// <summary>
58     /// <para>Reads and returns the first <typeparamref name="T"/> structure value from the
59     ///     ↳ file at the <paramref name="path"/>.</para>
60     /// <para>Считывает и возвращает первое значение структуры типа <typeparamref name="T"/>
61     ///     ↳ из файла находящегося в <paramref name="path"/>.</para>
62     /// </summary>
63     /// <typeparam name="T">
64     /// <para>The structure type.</para>
65     /// <para>Тип структуры.</para>
66     /// </typeparam>
67     /// <param name="path">
68     /// <para>The path to the file, from which to read the <typeparamref name="T"/>
69     ///     ↳ structure value.</para>
70     /// <para>Путь к файлу, из которого нужно прочитать значение структуры типа
71     ///     ↳ <typeparamref name="T"/>.</para>
72     /// </param>
73     /// <returns>
74     /// <para>The <typeparamref name="T"/> structure value if read from the file at the
75     ///     ↳ <paramref name="path"/> is successful; otherwise the default <typeparamref
76     ///     ↳ name="T"/> structure value.</para>
77     /// <para>Значение структуры типа <typeparamref name="T"/> если чтение из файла
78     ///     ↳ находящегося в <paramref name="path"/> прошло успешно, иначе значение структуры типа
79     ///     ↳ <typeparamref name="T"/> по умолчанию.</para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     public static T ReadFirstOrDefault<T>(string path)
83     where T : struct
84     {
85         using var fileStream = GetValidFileStreamOrDefault<T>(path);
86         return fileStream?.ReadOrDefault<T>() ?? default;
87     }
88
89     /// <summary>
90     /// <para>Returns the <see cref="FileStream"/> opened for reading from the file at the
91     ///     ↳ <paramref name="path"/> if the file exists, not empty and its size is a multiple of
92     ///     ↳ the <typeparamref name="TStruct"/> structure size; otherwise <see
93     ///     ↳ langword="null"/>.</para>

```

```

79  /// <para>Возвращает <see cref="FileStream"/> открытый для чтения из файла находящегося
    ↳ в <paramref name="path"/>, если файл существует, не пуст и его размер кратен размеру
    ↳ структуры типа <typeparamref name="TStruct"/>, а иначе <see langword="null"/>.</para>
80  /// </summary>
81  /// <typeparam name="TStruct">
82  /// <para>The structure type.</para>
83  /// <para>Тип структуры.</para>
84  /// </typeparam>
85  /// <param name="path">
86  /// <para>The path to the file to validate.</para>
87  /// <para>Путь к проверяемому файлу.</para>
88  /// </param>
89  /// <returns>
90  /// <para>A <see cref="FileStream"/> opened for reading in the case of successful check;
    ↳ otherwise <see langword="null"/>.</para>
91  /// <para><see cref="FileStream"/> открытый для чтения в случае успешной проверки, а
    ↳ иначе <see langword="null"/>.</para>
92  /// </returns>
93  /// <exception cref="InvalidOperationException">
94  /// <para>The size of the file at the <paramref name="path"/> is not a multiple of the
    ↳ required <typeparamref name="TStruct"/> structure size.</para>
95  /// <para>Размер файла находящегося в <paramref name="path"/> не кратен требуемому
    ↳ размеру структуры типа <typeparamref name="TStruct"/>.</para>
96  /// </exception>
97  [MethodImpl(MethodImplOptions.AggressiveInlining)]
98  private static FileStream GetValidFileStreamOrDefault(TStruct>(string path) where
    ↳ TStruct : struct => GetValidFileStreamOrDefault(path, Structure<TStruct>.Size);
99
100  /// <summary>
101  /// <para>Returns the <see cref="FileStream"/> opened for reading from the file at the
    ↳ <paramref name="path"/> if the file exists, not empty and its size is a multiple of
    ↳ the required <paramref name="elementSize"/>; otherwise <see langword="null"/>.</para>
102  /// <para>Возвращает <see cref="FileStream"/> открытый для чтения из файла находящегося
    ↳ в <paramref name="path"/>, если файл существует, не пуст и его размер кратен
    ↳ <paramref name="elementSize"/>, а иначе <see langword="null"/>.</para>
103  /// </summary>
104  /// <param name="path">
105  /// <para>The path to the file to validate.</para>
106  /// <para>Путь к проверяемому файлу.</para>
107  /// </param>
108  /// <param name="elementSize">
109  /// <para>Required size of elements located in the file at the <paramref
    ↳ name="path"/>.</para>
110  /// <para>Требуемый размер элементов, находящихся в файле находящегося в <paramref
    ↳ name="path"/>.</para>
111  /// </param>
112  /// <returns>
113  /// <para>A <see cref="FileStream"/> opened for reading in the case of successful check;
    ↳ otherwise <see langword="null"/>.</para>
114  /// <para><see cref="FileStream"/> открытый для чтения в случае успешной проверки, а
    ↳ иначе <see langword="null"/>.</para>
115  /// </returns>
116  /// <exception cref="InvalidOperationException">
117  /// <para>The size of the file at the <paramref name="path"/> is not a multiple of the
    ↳ required <paramref name="elementSize"/>.</para>
118  /// <para>Размер файла находящегося в <paramref name="path"/> не кратен требуемому
    ↳ <paramref name="elementSize"/>.</para>
119  /// </exception>
120  [MethodImpl(MethodImplOptions.AggressiveInlining)]
121  private static FileStream GetValidFileStreamOrDefault(string path, int elementSize)
122  {
123      if (!File.Exists(path))
124      {
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
            ↳ {elementSize}.");
131      }
132      return fileSize > 0 ? File.OpenRead(path) : null;
133  }
134
135  /// <summary>

```

```

136 /// <para>Reads and returns the last <typeparamref name="T"/> structure value from the
137   ↳ file at the <paramref name="path"/>.</para>
138 /// <para>Считывает и возвращает последнее значение структуры типа <typeparamref
139   ↳ name="T"/> из файла находящегося в <paramref name="path"/>.</para>
140 /// </summary>
141 /// <typeparam name="T">
142 /// <para>The structure type.</para>
143 /// <para>Тип структуры.</para>
144 /// </typeparam>
145 /// <param name="path">
146 /// <para>The path to the <typeparamref name="T"/> structure values.</para>
147 /// <para>Путь к файлу с значениями структур типа <typeparamref name="T"/>.</para>
148 /// </param>
149 /// <returns>
150 /// <para>The <typeparamref name="T"/> structure value from the file at the <paramref
151   ↳ name="path"/> in the case of successful read; otherwise the default <typeparamref
152   ↳ name="T"/> structure value.</para>
153 /// <para>Значение структуры типа <typeparamref name="T"/> из файла находящегося в
154   ↳ <paramref name="path"/> в случае успешного чтения, иначе значение по умолчанию
155   ↳ структуры типа <typeparamref name="T"/>.</para>
156 /// </returns>
157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 public static T ReadLastOrDefault<T>(string path)
159     where T : struct
160 {
161     var elementSize = Structure<T>.Size;
162     using var reader = GetValidFileStreamOrDefault(path, elementSize);
163     if (reader == null)
164     {
165         return default;
166     }
167     var totalElements = reader.Length / elementSize;
168     reader.Position = (totalElements - 1) * elementSize; // Set to last element
169     return reader.ReadOrDefault<T>();
170 }
171
172 /// <summary>
173 /// <para>Writes <typeparamref name="T"/> structure value at the beginning of the file
174   ↳ at the <paramref name="path"/>.</para>
175 /// <para>Записывает значение структуры типа <typeparamref name="T"/> в начало файла
176   ↳ находящегося в <paramref name="path"/>.</para>
177 /// </summary>
178 /// <typeparam name="T">
179 /// <para>The structure type.</para>
180 /// <para>Тип структуры.</para>
181 /// </typeparam>
182 /// <param name="path">
183 /// <para>The path to the file to be changed or created.</para>
184 /// <para>Путь к файлу, который будет изменён или создан.</para>
185 /// </param>
186 /// <param name="value">
187 /// <para><typeparamref name="T"/> structure value to be written at the beginning of the
188   ↳ file at the <paramref name="path"/>.</para>
189 /// <para>Значение структуры типа <typeparamref name="T"/>, записываемое в начало файла
190   ↳ находящегося в <paramref name="path"/>.</para>
191 /// </param>
192 [MethodImpl(MethodImplOptions.AggressiveInlining)]
193 public static void WriteFirst<T>(string path, T value)
194     where T : struct
195 {
196     using var writer = File.OpenWrite(path);
197     writer.Position = 0;
198     writer.Write(value);
199 }
200
201 /// <summary>
202 /// <para>Opens or creates the file at the <paramref name="path"/> and returns its <see
203   ↳ cref="FileStream"/> with append mode and write access.</para>
204 /// <para>Открывает или создаёт файл находящийся в <paramref name="path"/> и возвращает
205   ↳ его <see cref="FileStream"/> с режимом дополнения и доступом на запись.</para>
206 /// </summary>
207 /// <param name="path">
208 /// <para>The path to the file to open or create.</para>
209 /// <para>Путь к файлу, который нужно открыть или создать.</para>
210 /// </param>
211 /// <returns>
212 /// <para>The <see cref="FileStream"/> with append mode and write access.</para>

```

```

201 /// <para><see cref="FileStream"/> с режимом дополнения и доступом на запись.</para>
202 /// </returns>
203 [MethodImpl(MethodImplOptions.AggressiveInlining)]
204 public static FileStream Append(string path) => File.Open(path, FileMode.Append,
    ↳ FileAccess.Write);
205
206 /// <summary>
207 /// <para>Returns the size of file at the <paramref name="path"/> if file exists;
    ↳ otherwise 0.</para>
208 /// <para>Возвращает размер файла находящегося в <paramref name="path"/> если тот
    ↳ существует, иначе 0.</para>
209 /// </summary>
210 /// <param name="path">
211 /// <para>The path to the file to get size.</para>
212 /// <para>Путь к файлу, размер которого нужно получить.</para>
213 /// </param>
214 /// <returns>
215 /// <para>Size of file at the <paramref name="path"/> if it exists; otherwise 0.</para>
216 /// <para>Размер файла если файл находящийся в <paramref name="path"/> существует, иначе
    ↳ 0.</para>
217 /// </returns>
218 [MethodImpl(MethodImplOptions.AggressiveInlining)]
219 public static long GetSize(string path) => File.Exists(path) ? new FileInfo(path).Length
    ↳ : 0;
220
221 /// <summary>
222 /// <para>Sets the <paramref name="size"/> for the file at the <paramref
    ↳ name="path"/>.</para>
223 /// <para>Устанавливает <paramref name="size"/> файлу находящемуся в <paramref
    ↳ name="path"/>.</para>
224 /// </summary>
225 /// <param name="path">
226 /// <para>The path to the file to be resized.</para>
227 /// <para>Путь к файлу, размер которого нужно изменить.</para>
228 /// </param>
229 /// <param name="size">
230 /// <para>The size to assign to the file at the <paramref name="path"/>.</para>
231 /// <para>Размер который будет присвоен файлу находящемуся в <paramref
    ↳ name="path"/>.</para>
232 /// </param>
233 [MethodImpl(MethodImplOptions.AggressiveInlining)]
234 public static void SetSize(string path, long size)
235 {
236     using var fileStream = File.Open(path, FileMode.OpenOrCreate);
237     if (fileStream.Length != size)
238     {
239         fileStream.SetLength(size);
240     }
241 }
242
243 /// <summary>
244 /// <para>Removes all files from the directory at the path <paramref
    ↳ name="directory"/>.</para>
245 /// <para>Удаляет все файлы из директории находящейся по пути <paramref
    ↳ name="directory"/>.</para>
246 /// </summary>
247 /// <param name="directory">
248 /// <para>The path to the directory to be cleaned.</para>
249 /// <para>Путь к директории для очистки.</para>
250 /// </param>
251 [MethodImpl(MethodImplOptions.AggressiveInlining)]
252 public static void DeleteAll(string directory) => DeleteAll(directory, "*");
253
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>A 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>A 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>A <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 }

```

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"/> <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"/>
    ↳ 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>A 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>A 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>A 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.Tests/FileHelpersTests.cs

```

1  using System.IO;
2  using Xunit;
3
4  namespace Platform.IO.Tests
5  {
6      public class FileHelpersTests
7      {
8          [Fact]
9          public void WriteReadTest()
10         {
11             var temporaryFile = Path.GetTempFileName();
12             var originalValue = 42UL;
13             FileHelpers.WriteFirst(temporaryFile, originalValue);
14             var readValue = FileHelpers.ReadFirstOrDefault<ulong>(temporaryFile);
15             Assert.Equal(readValue, originalValue);
16             File.Delete(temporaryFile);
17         }
18     }
19 }

```

Index

- ./csharp/Platform.IO.Tests/FileHelpersTests.cs, 11
- ./csharp/Platform.IO/ConsoleCancellation.cs, 1
- ./csharp/Platform.IO/ConsoleHelpers.cs, 2
- ./csharp/Platform.IO/FileHelpers.cs, 4
- ./csharp/Platform.IO/StreamExtensions.cs, 9