

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     public class ConsoleCancellation : DisposableBase
12     {
13         public CancellationTokenSource Source
14         {
15             [MethodImpl(MethodImplOptions.AggressiveInlining)]
16             get;
17         }
18
19         public CancellationToken Token
20         {
21             [MethodImpl(MethodImplOptions.AggressiveInlining)]
22             get;
23         }
24
25         public bool IsRequested
26         {
27             [MethodImpl(MethodImplOptions.AggressiveInlining)]
28             get => Source.IsCancellationRequested;
29         }
30
31         public bool NotRequested
32         {
33             [MethodImpl(MethodImplOptions.AggressiveInlining)]
34             get => !Source.IsCancellationRequested;
35         }
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public ConsoleCancellation()
39         {
40             Source = new CancellationTokenSource();
41             Token = Source.Token;
42             Console.CancelKeyPress += OnCancelKeyPress;
43         }
44
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public void ForceCancellation() => Source.Cancel();
47
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public void Wait()
50         {
51             while (NotRequested)
52             {
53                 ThreadHelpers.Sleep();
54             }
55         }
56
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         protected override void Dispose(bool manual, bool wasDisposed)
59         {
60             if (!wasDisposed)
61             {
62                 Console.CancelKeyPress -= OnCancelKeyPress;
63                 Source.DisposeIfPossible();
64             }
65         }
66
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         private void OnCancelKeyPress(object sender, ConsoleCancelEventArgs e)
69         {
70             e.Cancel = true;
71             if (NotRequested)
72             {
73                 Source.Cancel();
74             }
75         }
76     }
77 }
```

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 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.IO
10 {
11     public static class ConsoleHelpers
12     {
13         /// <summary>
14         /// <para>Requests and expects a user to press any key in the console.</para>
15         /// <para>Запрашивает и ожидает нажатие любой клавиши пользователем в консоли.</para>
16         /// </summary>
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public static void PressAnyKeyToContinue()
19         {
20             Console.WriteLine("Press any key to continue.");
21             Console.ReadKey();
22         }
23
24         /// <summary>
25         /// <para>Gets the argument's value with the specified <paramref name="index" /> from
26         → the <paramref name="args"/> array and if it's absent requests a user to input it in
27         → the console.</para>
28         /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива
29         → <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у
30         → пользователя.</para>
31         /// </summary>
32         /// <param name="index">
33         /// <para>The ordinal number of the argument in the array.</para>
34         /// <para>Порядковый номер аргумента в массиве.</para>
35         /// </param>
36         /// <param name="args">
37         /// <para>The argument array passed to the application.</para>
38         /// <para>Массив аргументов переданных приложению.</para>
39         /// </param>
40         /// <returns>
41         /// <para>The value with the specified <paramref name="index"/> extracted from the
42         → <paramref name="args"/> array or entered by a user in the console.</para>
43         /// <para>Значение с указанным <paramref name="index"/>, извлечённое из массива
44         → <paramref name="args"/>, или введённое пользователем в консоли.</para>
45         /// </returns>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static string GetOrReadArgument(int index, params string[] args) =>
48         → GetOrReadArgument(index, $"{index + 1} argument", args);
49
50         /// <summary>
51         /// <para>Gets the argument's value with the specified <paramref name="index" /> from
52         → the <paramref name="args"/> array and if it's absent requests a user to input it in
53         → the console.</para>
54         /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива
55         → <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у
56         → пользователя.</para>
57         /// </summary>
58         /// <param name="index">
59         /// <para>The ordinal number of the argument in the array.</para>
60         /// <para>Порядковый номер аргумента в массиве.</para>
61         /// </param>
62         /// <param name="readMessage">
63         /// <para>The message's text to a user describing which argument is being entered at the
64         → moment. If the <paramref name="args"/> array doesn't contain the element with the
65         → specified <paramref name="index"/>, then this message is used.</para>
66         /// <para>Текст сообщения пользователю описывающее какой аргумент вводится в данный
67         → момент. Это сообщение используется только если массив <paramref name="args"/> не
68         → содержит аргумента с указанным <paramref name="index"/>.</para>
69         /// </param>
70         /// <param name="args">
71         /// <para>The argument array passed to the application.</para>
72         /// <para>Массив аргументов переданных приложению.</para>
73         /// </param>
74         /// <returns>
75         /// <para>The value with the specified <paramref name="index"/> extracted from the
76         → <paramref name="args"/> array or entered by a user in the console.</para>
```

```

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

```

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  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7

```

```

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

```

```

71 [MethodImpl(MethodImplOptions.AggressiveInlining)]
72 public static T ReadFirstOrDefault<T>(string path)
73     where T : struct
74 {
75     using var fileStream = GetValidFileStreamOrDefault<T>(path);
76     return fileStream?.ReadOrDefault<T>() ?? default;
77 }
78
79 /// <summary>
80 /// <para>Returns the <see cref="FileStream"/> opened for reading from the file at the
81     ↳ <paramref name="path"/> if the file exists, not empty and its size is a multiple of
82     ↳ the <typeparamref name="TStruct"/> structure size; otherwise <see
83     ↳ langword="null"/>.</para>
84 /// <para>Возвращает <see cref="FileStream"/> открытый для чтения из файла находящегося
85     ↳ в <paramref name="path"/>, если файл существует, не пуст и его размер кратен размеру
86     ↳ структуры типа <typeparamref name="TStruct"/>, а иначе <see langword="null"/>.</para>
87 /// </summary>
88 /// <typeparamref name="TStruct">
89 /// <para>The structure type.</para>
90 /// <para>Тип структуры.</para>
91 /// </typeparamref>
92 /// <param name="path">
93 /// <para>The path to the file to validate.</para>
94 /// <para>Путь к проверяемому файлу.</para>
95 /// </param>
96 /// <returns>
97 /// <para>A <see cref="FileStream"/> opened for reading in the case of successful check;
98     ↳ otherwise <see langword="null"/>.</para>
99 /// <para><see cref="FileStream"/> открытый для чтения в случае успешной проверки, а
100     ↳ иначе <see langword="null"/>.</para>
101 /// </returns>
102 /// <exception cref="InvalidOperationException">
103 /// <para>The size of the file at the <paramref name="path"/> is not a multiple of the
104     ↳ required <paramref name="elementSize"/>.</para>
105 /// <para>Размер файла находящегося в <paramref name="path"/> не кратен требуемому
106     ↳ <paramref name="elementSize"/>.</para>
107 /// </exception>
108 [MethodImpl(MethodImplOptions.AggressiveInlining)]
109 private static FileStream GetValidFileStreamOrDefault<TStruct>(string path) where
110     ↳ TStruct : struct => GetValidFileStreamOrDefault(path, Structure<TStruct>.Size);
111
112 /// <summary>
113 /// <para>Returns the <see cref="FileStream"/> opened for reading from the file at the
114     ↳ <paramref name="path"/> if the file exists, not empty and its size is a multiple of
115     ↳ the required <paramref name="elementSize"/>; otherwise <see langword="null"/>.</para>
116 /// <para>Возвращает <see cref="FileStream"/> открытый для чтения из файла находящегося
117     ↳ в <paramref name="path"/>, если файл существует, не пуст и его размер кратен
118     ↳ <paramref name="elementSize"/>, а иначе <see langword="null"/>.</para>
119 /// </summary>
120 /// <param name="path">
121 /// <para>The path to the file to validate.</para>
122 /// <para>Путь к проверяемому файлу.</para>
123 /// </param>
124 /// <param name="elementSize">
125 /// <para>Required size of elements located in the file at the <paramref
126     ↳ name="path"/>.</para>
127 /// <para>Требуемый размер элементов, находящихся в файле находящегося в <paramref
128     ↳ name="path"/>.</para>
129 /// </param>
130 /// <returns>
131 /// <para>A <see cref="FileStream"/> opened for reading in the case of successful check;
132     ↳ otherwise <see langword="null"/>.</para>
133 /// <para><see cref="FileStream"/> открытый для чтения в случае успешной проверки, а
134     ↳ иначе <see langword="null"/>.</para>
135 /// </returns>
136 /// <exception cref="InvalidOperationException">
137 /// <para>The size of the file at the <paramref name="path"/> is not a multiple of the
138     ↳ required <paramref name="elementSize"/>.</para>
139 /// <para>Размер файла находящегося в <paramref name="path"/> не кратен требуемому
140     ↳ <paramref name="elementSize"/>.</para>
141 /// </exception>
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 private static FileStream GetValidFileStreamOrDefault(string path, int elementSize)
144 {
145     if (!File.Exists(path))
146     {

```

```

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

```

```

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

```

```

257     /// <para>Removes files from the directory at the path <paramref name="directory"/>
    → according to the <paramref name="searchPattern"/>.</para>
258     /// <para>Удаляет файлы из директории находящейся по пути <paramref name="directory"/> в
    → соответствии с <paramref name="searchPattern"/>.</para>
259     /// </summary>
260     /// <param name="directory">
261     /// <para>The path to the directory to be cleaned.</para>
262     /// <para>Путь к директории для очистки.</para>
263     /// </param>
264     /// <param name="searchPattern">
265     /// <para>A search pattern for files to be deleted in the directory at the path
    → <paramref name="directory"/>.</para>
266     /// <para>Шаблон поиска для удаляемых файлов в директории находящейся по пути <paramref
    → name="directory"/>.</para>
267     /// </param>
268     [MethodImpl(MethodImplOptions.AggressiveInlining)]
269     public static void DeleteAll(string directory, string searchPattern) =>
    → DeleteAll(directory, searchPattern, SearchOption.TopDirectoryOnly);
270
271     /// <summary>
272     /// <para>Removes files from the directory at the path <paramref name="directory"/>
    → according to the <paramref name="searchPattern"/> and the <paramref
    → name="searchOption"/>.</para>
273     /// <para>Удаляет файлы из директории находящейся по пути <paramref name="directory"/> в
    → соответствии с <paramref name="searchPattern"/> и <paramref
    → name="searchOption"/>.</para>
274     /// </summary>
275     /// <param name="directory">
276     /// <para>The path to the directory to be cleaned.</para>
277     /// <para>Путь к директории для очистки.</para>
278     /// </param>
279     /// <param name="searchPattern">
280     /// <para>A search pattern for files to be deleted in the directory at the path
    → <paramref name="directory"/>.</para>
281     /// <para>Шаблон поиска для удаляемых файлов в директории находящейся по пути <paramref
    → name="directory"/> .</para>
282     /// </param>
283     /// <param name="searchOption">
284     /// <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>
285     /// <para>Значение <see cref="SearchOption"/> определяющее искать ли только в текущей
    → директории находящейся по пути <paramref name="directory"/>, или также во всех
    → субдиректориях.</para>
286     /// </param>
287     [MethodImpl(MethodImplOptions.AggressiveInlining)]
288     public static void DeleteAll(string directory, string searchPattern, SearchOption
    → searchOption)
289     {
290         foreach (var file in Directory.EnumerateFiles(directory, searchPattern,
    → searchOption))
291         {
292             File.Delete(file);
293         }
294     }
295 }
296 }

```

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

```

1  using System.IO;
2  using System.Runtime.CompilerServices;
3  using Platform.Unsafe;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.IO
8  {
9      /// <summary>
10     /// <para>Provides a set of extension methods for <see cref="Stream"/> class
    → instances.</para>
11     /// <para>Предоставляет набор методов расширения для экземпляров класса <see
    → cref="Stream"/>.</para>
12     /// </summary>
13     public static class StreamExtensions
14     {
15         /// <summary>

```



```

16  /// <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>
17  /// <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>
18  /// </summary>
19  /// <typeparam name="T">
20  /// <para>The structure type.</para>
21  /// <para>Тип структуры.</para>
22  /// </typeparam>
23  /// <param name="stream">
24  /// <para>A stream to write to.</para>
25  /// <para>Поток, в который осуществляется запись.</para>
26  /// </param>
27  /// <param name="value">
28  /// <para>The <typeparam name="T"> structure value to be written to the <paramref
    → name="stream"/>.</para>
29  /// <para>Значение структуры типа <typeparam name="T"> которое будет записано в поток
    → <paramref name="stream"/>.</para>
30  /// </param>
31  [MethodImpl(MethodImplOptions.AggressiveInlining)]
32  public static void Write<T>(this Stream stream, T value)
33      where T : struct
34  {
35      var bytes = value.ToBytes();
36      stream.Write(bytes, 0, bytes.Length);
37  }
38
39  /// <summary>
40  /// <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>
41  /// <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>
42  /// </summary>
43  /// <typeparam name="T">
44  /// <para>The structure type.</para>
45  /// <para>Тип структуры.</para>
46  /// </typeparam>
47  /// <param name="stream">
48  /// <para>A stream containing the <typeparam name="T"> structure value.</para>
49  /// <para>Поток, содержащий значение структуры типа <typeparam name="T">.</para>
50  /// </param>
51  /// <returns>
52  /// <para>The <typeparam name="T"> structure value, if its bytes from the <paramref
    → name="stream"/> are read; otherwise the default <typeparamref name="T"/> structure
    → value.</para>
53  /// <para>Значение структуры типа <typeparam name="T">, если её байты из потока
    → <paramref name="stream"/> были прочитаны, иначе значение структуры типа
    → <typeparamref name="T"/> по умолчанию.</para>
54  /// </returns>
55  [MethodImpl(MethodImplOptions.AggressiveInlining)]
56  public static T ReadOrDefault<T>(this Stream stream)
57      where T : struct
58  {
59      var size = Structure<T>.Size;
60      var buffer = new byte[size];
61      return stream.Read(buffer, 0, size) == size ? buffer.ToStructure<T>() : default;
62  }
63
64  /// <summary>
65  /// <para>Reads and returns all <typeparam name="T"> structure values array from the
    → <paramref name="stream"/>.</para>
66  /// <para>Прочитывает и возвращает массив всех значений структур типа <typeparam
    → name="T"> из потока <paramref name="stream"/>.</para>
67  /// </summary>
68  /// <typeparam name="T">
69  /// <para>The structure type.</para>
70  /// <para>Тип структуры.</para>
71  /// </typeparam>
72  /// <param name="stream">
73  /// <para>A stream containing the <typeparam name="T"> structure values.</para>
74  /// <para>Поток, содержащий значения структур типа <typeparam name="T">.</para>
75  /// </param>
76  /// <returns>

```

```

77     /// <para>The <typeparam name="T"> structure values array read from the <paramref
    ↪     name="stream"/>.</para>
78     /// <para>Массив с значениями структур типа <typeparam name="T">, прочитанными из потока
    ↪     <paramref name="stream"/>.</para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public static T[] ReadAll<T>(this Stream stream)
82     where T : struct
83     {
84         var size = Structure<T>.Size;
85         var buffer = new byte[size];
86         var elementsLength = stream.Length / size;
87         var elements = new T[elementsLength];
88         for (var i = 0; i < elementsLength; i++)
89         {
90             stream.Read(buffer, 0, size);
91             elements[i] = buffer.ToStructure<T>();
92         }
93         return elements;
94     }
95 }
96 }

```

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, 10
- ./csharp/Platform.IO/ConsoleCancellation.cs, 1
- ./csharp/Platform.IO/ConsoleHelpers.cs, 1
- ./csharp/Platform.IO/FileHelpers.cs, 3
- ./csharp/Platform.IO/StreamExtensions.cs, 8