

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    [MethodImpl(MethodImplOptions.AggressiveInlining)]
117    private void OnCancelKeyPress(object sender, ConsoleCancelEventArgs e)
118    {
119        e.Cancel = true;
120        if (NotRequested)
121        {
122            Source.Cancel();
123        }
124    }
125    }

```

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  {
51      GetOrReadArgument(index, $"{index + 1} argument", args);
52  }
53
54  /// <summary>
55  /// <para>Gets an argument's value with the specified <paramref name="index"/> from the
56  ///     <paramref name="args"/> array and if it's absent requests a user to input it in the
57  ///     console.</para>
58  /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива
59  ///     <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у
60  ///     пользователя.</para>
61  /// </summary>
62  /// <param name="index">
63  /// <para>The ordinal number of the argument in the array.</para>
64  /// <para>Порядковый номер аргумента в массиве.</para>
65  /// </param>
66  /// <param name="readMessage">
67  /// <para>The message's text to a user describing which argument is being entered at the
68  ///     moment. If the <paramref name="args"/> array doesn't contain the element with the
69  ///     specified <paramref name="index"/>, then this message is used.</para>
70  /// <para>Текст сообщения пользователю описывающее какой аргумент вводится в данный
71  ///     момент. Это сообщение используется только если массив <paramref name="args"/> не
72  ///     содержит аргумента с указанным <paramref name="index"/>.</para>
73  /// </param>
74  /// <param name="args">
75  /// <para>The argument array passed to the application.</para>
76  /// <para>Массив аргументов переданных приложению.</para>
77  /// </param>
78  /// <returns>
79  /// <para>The value with the specified <paramref name="index"/> extracted from the
80  ///     <paramref name="args"/> array or entered by a user in the console.</para>
81  /// <para>Значение с указанным <paramref name="index"/>, извлечённое из массива
82  ///     <paramref name="args"/>, или введённое пользователем в консоли.</para>
83  /// </returns>
84  [MethodImpl(MethodImplOptions.AggressiveInlining)]
85  public static string GetOrReadArgument(int index, string readMessage, params string[]
86  {
87      args)
88  {
89      if (!args.TryGetElement(index, out string result))
90      {
91          Console.Write($"{readMessage}: ");
92          result = Console.ReadLine();
93      }
94      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 если приложение было скомпилировано с директивой
95   → 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 to the standard output stream using the specified <paramref name="format"/>,
103   → followed by the current line terminator.</para>
104 /// <para>Записывает текстовые представления объектов заданного массива <paramref
105   → name="args"/>, в стандартный выходной поток с использованием заданного <paramref
106   → name="format"/>, за которым следует текущий признак конца строки.</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 если приложение было скомпилировано с директивой
122   → 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 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">

```

```

19  /// <para>The path to a 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 a 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 a file at
31  ///   ↳ 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 a 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 a
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 a file, from which to read the first <typeparamref name="T"/>
69  ///   ↳ structure value.</para>
70  /// <para>Путь к файлу, из которого нужно прочитать значение первой структуры типа
71  ///   ↳ <typeparamref name="T"/>.</para>
72  /// </param>
73  /// <returns>
74  /// <para>The <typeparamref name="T"/> structure value if read from a 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  [MethodImpl(MethodImplOptions.AggressiveInlining)]
90  private static FileStream GetValidFileStreamOrDefault<TStruct>(string path) where
91  ///   ↳ TStruct : struct => GetValidFileStreamOrDefault(path, Structure<TStruct>.Size);
92  [MethodImpl(MethodImplOptions.AggressiveInlining)]
93  private static FileStream GetValidFileStreamOrDefault(string path, int elementSize)
94  {
95      if (!File.Exists(path))
96      {

```

```

83         return null;
84     }
85     var fileSize = GetSize(path);
86     if (fileSize % elementSize != 0)
87     {
88         throw new InvalidOperationException($"File is not aligned to elements with size
89         ↳ {elementSize}.");
90     }
91     return fileSize > 0 ? File.OpenRead(path) : null;
92 }
93
94 /// <summary>
95 /// <para>Reads and returns the last <typeparamref name="T"/> structure value from a
96 ↳ file at the <paramref name="path"/>.</para>
97 /// <para>Считывает и возвращает последнее значение структуры типа <typeparamref
98 ↳ name="T"/> из файла находящегося в <paramref name="path"/>.</para>
99 /// </summary>
100 /// <typeparam name="T">
101 /// <para>The structure type.</para>
102 /// <para>Тип структуры.</para>
103 /// </typeparam>
104 /// <param name="path">
105 /// <para>The path to a file, from which to read the last <typeparamref name="T"/>
106 ↳ structure value.</para>
107 /// <para>Путь к файлу, из которого нужно прочесть значение последней структуры типа
108 ↳ <typeparamref name="T"/>.</para>
109 /// </param>
110 /// <returns>
111 /// <para>The <typeparamref name="T"/> structure value if read from a file at the
112 ↳ <paramref name="path"/> is successful; otherwise the default <typeparamref
113 ↳ name="T"/> structure value.</para>
114 /// <para>Значение структуры типа <typeparamref name="T"/> из файла находящегося в
115 ↳ <paramref name="path"/> в случае успешного чтения, иначе значение по умолчанию
116 ↳ структуры типа <typeparamref name="T"/>.</para>
117 /// </returns>
118 [MethodImpl(MethodImplOptions.AggressiveInlining)]
119 public static T ReadLastOrDefault<T>(string path)
120     where T : struct
121 {
122     var elementSize = Structure<T>.Size;
123     using var reader = GetValidFileStreamOrDefault(path, elementSize);
124     if (reader == null)
125     {
126         return default;
127     }
128     var totalElements = reader.Length / elementSize;
129     reader.Position = (totalElements - 1) * elementSize; // Set to last element
130     return reader.ReadOrDefault<T>();
131 }
132
133 /// <summary>
134 /// <para>Writes <typeparamref name="T"/> structure value at the beginning of a file at
135 ↳ the <paramref name="path"/>.</para>
136 /// <para>Записывает значение структуры типа <typeparamref name="T"/> в начало файла
137 ↳ находящегося в <paramref name="path"/>.</para>
138 /// </summary>
139 /// <typeparam name="T">
140 /// <para>The structure type.</para>
141 /// <para>Тип структуры.</para>
142 /// </typeparam>
143 /// <param name="path">
144 /// <para>The path to a file to be changed or created.</para>
145 /// <para>Путь к файлу, который будет изменён или создан.</para>
146 /// </param>
147 /// <param name="value">
148 /// <para>The <typeparamref name="T"/> structure value to be written at the beginning of
149 ↳ a file at the <paramref name="path"/>.</para>
150 /// <para>Значение структуры типа <typeparamref name="T"/>, записываемое в начало файла
151 ↳ находящегося в <paramref name="path"/>.</para>
152 /// </param>
153 [MethodImpl(MethodImplOptions.AggressiveInlining)]
154 public static void WriteFirst<T>(string path, T value)
155     where T : struct
156 {
157     using var writer = File.OpenWrite(path);
158     writer.Position = 0;
159     writer.Write(value);
160 }

```

```

147 }
148
149 /// <summary>
150 /// <para>Opens or creates a file at the <paramref name="path"/> and returns its <see
    ↳ cref="FileStream"/> with append mode and write access.</para>
151 /// <para>Открывает или создаёт файл находящийся в <paramref name="path"/> и возвращает
    ↳ его <see cref="FileStream"/> с режимом дополнения и доступом на запись.</para>
152 /// </summary>
153 /// <param name="path">
154 /// <para>The path to a file to open or create.</para>
155 /// <para>Путь к файлу, который нужно открыть или создать.</para>
156 /// </param>
157 /// <returns>
158 /// <para>The <see cref="FileStream"/> with append mode and write access.</para>
159 /// <para><see cref="FileStream"/> с режимом дополнения и доступом на запись.</para>
160 /// </returns>
161 [MethodImpl(MethodImplOptions.AggressiveInlining)]
162 public static FileStream Append(string path) => File.Open(path, FileMode.Append,
    ↳ FileAccess.Write);
163
164 /// <summary>
165 /// <para>Returns the size of a file at the <paramref name="path"/> if the file exists;
    ↳ otherwise 0.</para>
166 /// <para>Возвращает размер файла находящегося в <paramref name="path"/> если тот
    ↳ существует, иначе 0.</para>
167 /// </summary>
168 /// <param name="path">
169 /// <para>The path to a file to get size.</para>
170 /// <para>Путь к файлу, размер которого нужно получить.</para>
171 /// </param>
172 /// <returns>
173 /// <para>Size of a file at the <paramref name="path"/> if it exists; otherwise 0.</para>
174 /// <para>Размер файла если файл находящийся в <paramref name="path"/> существует, иначе
    ↳ 0.</para>
175 /// </returns>
176 [MethodImpl(MethodImplOptions.AggressiveInlining)]
177 public static long GetSize(string path) => File.Exists(path) ? new FileInfo(path).Length
    ↳ : 0;
178
179 /// <summary>
180 /// <para>Sets the <paramref name="size"/> for a file at the <paramref
    ↳ name="path"/>.</para>
181 /// <para>Устанавливает <paramref name="size"/> файлу находящемуся по пути <paramref
    ↳ name="path"/>.</para>
182 /// </summary>
183 /// <param name="path">
184 /// <para>The path to a file to be resized.</para>
185 /// <para>Путь к файлу, размер которого нужно изменить.</para>
186 /// </param>
187 /// <param name="size">
188 /// <para>The size to assign to a file at the <paramref name="path"/>.</para>
189 /// <para>Размер который будет присвоен файлу находящемуся по пути <paramref
    ↳ name="path"/>.</para>
190 /// </param>
191 [MethodImpl(MethodImplOptions.AggressiveInlining)]
192 public static void SetSize(string path, long size)
193 {
194     using var fileStream = File.Open(path, FileMode.OpenOrCreate);
195     if (fileStream.Length != size)
196     {
197         fileStream.SetLength(size);
198     }
199 }
200
201 /// <summary>
202 /// <para>Removes all files from the directory at the path <paramref
    ↳ name="directory"/>.</para>
203 /// <para>Удаляет все файлы из директории находящейся по пути <paramref
    ↳ name="directory"/>.</para>
204 /// </summary>
205 /// <param name="directory">
206 /// <para>The path to the directory to be cleaned.</para>
207 /// <para>Путь к директории для очистки.</para>
208 /// </param>
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 public static void DeleteAll(string directory) => DeleteAll(directory, "*");
211

```

```

212 /// <summary>
213 /// <para>Removes files from the directory at the path <paramref name="directory"/>
    ↳ according to the <paramref name="searchPattern"/>.</para>
214 /// <para>Удаляет файлы из директории находящейся по пути <paramref name="directory"/> в
    ↳ соответствии с <paramref name="searchPattern"/>.</para>
215 /// </summary>
216 /// <param name="directory">
217 /// <para>The path to the directory to be cleaned.</para>
218 /// <para>Путь к директории для очистки.</para>
219 /// </param>
220 /// <param name="searchPattern">
221 /// <para>The search pattern for files to be deleted in the directory at the path
    ↳ <paramref name="directory"/>.</para>
222 /// <para>Шаблон поиска для удаляемых файлов в директории находящейся по пути <paramref
    ↳ name="directory"/>.</para>
223 /// </param>
224 [MethodImpl(MethodImplOptions.AggressiveInlining)]
225 public static void DeleteAll(string directory, string searchPattern) =>
    ↳ DeleteAll(directory, searchPattern, SearchOption.TopDirectoryOnly);
226
227 /// <summary>
228 /// <para>Removes files from the directory at the path <paramref name="directory"/>
    ↳ according to the <paramref name="searchPattern"/> and the <paramref
    ↳ name="searchOption"/>.</para>
229 /// <para>Удаляет файлы из директории находящейся по пути <paramref name="directory"/> в
    ↳ соответствии с <paramref name="searchPattern"/> и <paramref
    ↳ name="searchOption"/>.</para>
230 /// </summary>
231 /// <param name="directory">
232 /// <para>The path to the directory to be cleaned.</para>
233 /// <para>Путь к директории для очистки.</para>
234 /// </param>
235 /// <param name="searchPattern">
236 /// <para>The search pattern for files to be deleted in the directory at the path
    ↳ <paramref name="directory"/>.</para>
237 /// <para>Шаблон поиска для удаляемых файлов в директории находящейся по пути <paramref
    ↳ name="directory"/> .</para>
238 /// </param>
239 /// <param name="searchOption">
240 /// <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>
241 /// <para>Значение <see cref="SearchOption"/> определяющее искать ли только в текущей
    ↳ директории находящейся по пути <paramref name="directory"/>, или также во всех
    ↳ субдиректориях.</para>
242 /// </param>
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 public static void DeleteAll(string directory, string searchPattern, SearchOption
    ↳ searchOption)
245 {
246     foreach (var file in Directory.EnumerateFiles(directory, searchPattern,
    ↳ searchOption))
247     {
248         File.Delete(file);
249     }
250 }
251
252 /// <summary>
253 /// <para>Truncates the file at the <paramref name="path"/>.</para>
254 /// <para>Очищает содержимое файла по пути <paramref name="path"/>.</para>
255 /// </summary>
256 /// <param name="path">
257 /// <para>A path to a file to be truncated.</para>
258 /// <para>Путь к файлу для очистки содержимого.</para>
259 /// </param>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 public static void Truncate(string path) => File.Open(path, FileMode.Truncate).Dispose();
262
263 /// <summary>
264 /// <para>Appends the <paramref name="content"/> to a file at the <paramref
    ↳ name="path"/>.</para>
265 /// <para>Добавляет <paramref name="content"/> в конец файла по пути <paramref
    ↳ name="path"/>.</para>
266 /// </summary>
267 /// <param name="path">
268 /// <para>The path to a file to be appended by the <paramref name="content"/>.</para>

```



```

269     /// <para>Путь к файлу для добавления <paramref name="content"/> в конец файла.</para>
270     /// </param>
271     /// <param name="content">
272     /// <para>A content to be appended to a file at the file at the <paramref
    → name="path"/>.</para>
273     /// <para>Содержимое для добавления в конец файла по пути <paramref name="path"/>.</para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     public static void AppendLine(string path, string content)
277     {
278         using var writer = File.AppendText(path);
279         writer.WriteLine(content);
280     }
281
282     /// <summary>
283     /// <para>Performs the <paramref name="action"/> for each line of a file at the
    → <paramref name="path"/>.</para>
284     /// <para>Выполняет <paramref name="action"/> для каждой строки файла по пути <paramref
    → name="path"/>.</para>
285     /// </summary>
286     /// <param name="path">
287     /// <para>A path to a file to perform the <paramref name="action"/> for each line of the
    → file.</para>
288     /// <para>Путь к файлу для выполнения <paramref name="action"/> для каждой строки
    → файла.</para>
289     /// </param>
290     /// <param name="action">
291     /// <para>An action to be performed for each line of a file at the <paramref
    → name="path"/>.</para>
292     /// <para>Действие выполняемое для каждой строки файла по пути <paramref
    → name="path"/>.</para>
293     /// </param>
294     [MethodImpl(MethodImplOptions.AggressiveInlining)]
295     public static void EachLine(string path, Action<string> action)
296     {
297         using var reader = new StreamReader(path);
298         string line;
299         while ((line = reader.ReadLine()) != null)
300         {
301             action(line);
302         }
303     }
304 }
305 }

```

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        private const string UserFilesListFileNamePrefix = ".used-temporary-files.txt";
14        private static readonly object UsedFilesListLock = new();
15        private static readonly string UsedFilesListFilename =
16            ↪ Assembly.GetExecutingAssembly().Location + UserFilesListFileNamePrefix;
17        [MethodImpl(MethodImplOptions.AggressiveInlining)]
18        private static void AddToUsedFilesList(string filename)
19        {
20            lock (UsedFilesListLock)
21            {
22                FileHelpers.AppendLine(UsedFilesListFilename, filename);
23            }
24
25            /// <summary>
26            /// <para>Gets a temporary file and adds it to the used files list.</para>
27            /// <para>Получает временный файл и добавляет его в список использованных файлов.</para>
28            /// </summary>
29            /// <returns>
30            /// <para>The temporary file path.</para>
31            /// <para>Путь временного файла.</para>
32            /// </returns>
33            [MethodImpl(MethodImplOptions.AggressiveInlining)]
34            public static string UseNew()
35            {
36                var filename = Path.GetTempFileName();
37                AddToUsedFilesList(filename);
38                return filename;
39            }
40
41            /// <summary>
42            /// <para>Deletes all previously used temporary files and clears the files list.</para>
43            /// <para>Удаляет все ранее использованные временные файлы и очищает список
44            ↪ файлов.</para>
45            /// </summary>
46            [MethodImpl(MethodImplOptions.AggressiveInlining)]
47            public static void DeleteAllPreviouslyUsed()
48            {
49                lock (UsedFilesListLock)
50                {
51                    var listFilename = UsedFilesListFilename;
52                    if (File.Exists(listFilename))
53                    {
54                        FileHelpers.EachLine(listFilename, File.Delete);
55                        FileHelpers.Truncate(listFilename);
56                    }
57                }
58            }
59        }
60    }
```

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;
30             process.StartInfo.RedirectStandardOutput = true;
31             process.Start();
32             var path = process.StandardOutput.ReadLine();
33             Assert.True(File.Exists(path));
34             process.WaitForExit();
35             Assert.False(File.Exists(path));
36         }
37
38         /// <summary>
39         /// <para>
40         /// Tests that temporary file test without console app.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         [Fact]
45         public void TemporaryFileTestWithoutConsoleApp()
46         {
47             string fileName;
48             using (TemporaryFile tempFile = new())
49             {
50                 fileName = tempFile;
51                 Assert.True(File.Exists(fileName));
52             }
53             Assert.False(File.Exists(fileName));
54         }
55     }
56 }

```

Index

- ./csharp/Platform.IO.Tests/FileHelpersTests.cs, 12
- ./csharp/Platform.IO.Tests/TemporaryFileTests.cs, 13
- ./csharp/Platform.IO/ConsoleCancellation.cs, 1
- ./csharp/Platform.IO/ConsoleHelpers.cs, 2
- ./csharp/Platform.IO/FileHelpers.cs, 4
- ./csharp/Platform.IO/StreamExtensions.cs, 9
- ./csharp/Platform.IO/TemporaryFile.cs, 11
- ./csharp/Platform.IO/TemporaryFiles.cs, 12