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

```
using System;
using System.Runtime.CompilerServices;
using System.Threading;
using Platform.Disposables;
using Platform.Threading;

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

namespace Platform.IO
{
    public class ConsoleCancellation : DisposableBase
    {
        public CancellationTokenSource Source
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

        public CancellationToken Token
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

        public bool IsRequested
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => Source.IsCancellationRequested;
        }

        public bool NotRequested
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => !Source.IsCancellationRequested;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public ConsoleCancellation()
        {
            Source = new CancellationTokenSource();
            Token = Source.Token;
            Console.CancelKeyPress += OnCancelKeyPress;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void ForceCancellation() => Source.Cancel();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Wait()
        {
            while (NotRequested)
            {
                ThreadHelpers.Sleep();
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                Console.CancelKeyPress -= OnCancelKeyPress;
                Source.DisposeIfPossible();
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private void OnCancelKeyPress(object sender, ConsoleCancelEventArgs e)
        {
            e.Cancel = true;
            if (NotRequested)
            {
                Source.Cancel();
            }
        }
    }
}
```

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

```
using System;
using System.Diagnostics;
using System.Runtime.CompilerServices;
using Platform.Collections;
using Platform.Collections.Arrays;

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

namespace Platform.IO
{
    public static class ConsoleHelpers
    {
        /// <summary>
        /// <para>Requests and expects a user to press any key in the console.</para>
        /// <para>Запрашивает и ожидает нажатие любой клавиши пользователем в консоли.</para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void PressAnyKeyToContinue()
        {
            Console.WriteLine("Press any key to continue.");
            Console.ReadKey();
        }

        /// <summary>
        /// <para>Gets the argument's value with the specified <paramref name="index" /> from the <paramref name="args"/> array and if it's absent requests a user to input it in the console.</para>
        /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у пользователя.</para>
        /// </summary>
        /// <param name="index">
        /// <para>The ordinal number of the argument in the array.</para>
        /// <para>Порядковый номер аргумента в массиве.</para>
        /// </param>
        /// <param name="args">
        /// <para>The argument array passed to the application.</para>
        /// <para>Массив аргументов переданных приложению.</para>
        /// </param>
        /// <returns>
        /// <para>The value with the specified <paramref name="index"/> extracted from the <paramref name="args"/> array or entered by a user in the console.</para>
        /// <para>Значение с указанным <paramref name="index"/>, извлечённое из массива <paramref name="args"/>, или введённое пользователем в консоли.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string GetOrReadArgument(int index, params string[] args) => GetOrReadArgument(index, $"{index + 1} argument", args);

        /// <summary>
        /// <para>Gets the argument's value with the specified <paramref name="index" /> from the <paramref name="args"/> array and if it's absent requests a user to input it in the console.</para>
        /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у пользователя.</para>
        /// </summary>
        /// <param name="index">
        /// <para>The ordinal number of the argument in the array.</para>
        /// <para>Порядковый номер аргумента в массиве.</para>
        /// </param>
        /// <param name="readMessage">
        /// <para>The message's text to a user describing which argument is being entered at the moment. If the <paramref name="args"/> array doesn't contain the element with the specified <paramref name="index"/>, then this message is used.</para>
        /// <para>Текст сообщения пользователю описывающее какой аргумент вводится в данный момент. Это сообщение используется только если массив <paramref name="args"/> не содержит аргумента с указанным <paramref name="index"/>.</para>
        /// </param>
        /// <param name="args">
        /// <para>The argument array passed to the application.</para>
        /// <para>Массив аргументов переданных приложению.</para>
        /// </param>
        /// <returns>
        /// <para>The value with the specified <paramref name="index"/> extracted from the <paramref name="args"/> array or entered by a user in the console.</para>
```

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

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
```

```
namespace Platform.IO
{
    /// <summary>
    /// <para>Provides a set of helper methods to work with files.</para>
    /// <para>Предоставляет набор вспомогательных методов для работы с файлами.</para>
    /// </summary>
    public static class FileHelpers
    {
        /// <summary>
        /// <para>Reads all the text and returns character array from the file at the <paramref name="path"/>.</para>
        /// <para>Читает весь текст и возвращает массив символов из файла находящегося в <paramref name="path"/>.</para>
        /// </summary>
        /// <param name="path">
        /// <para>The path to the file, from which to read the character array.</para>
        /// <para>Путь к файлу, из которого нужно прочитать массив символов.</para>
        /// </param>
        /// <returns>
        /// <para>The character array from the file at the <paramref name="path"/>.</para>
        /// <para>Массив символов из файла находящегося в <paramref name="path"/>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static char[] ReadAllChars(string path) => File.ReadAllText(path).ToCharArray();

        /// <summary>
        /// <para>Reads and returns all <typeparamref name="T"/> structure values from the file at the <paramref name="path"/>.</para>
        /// <para>Считывает и возвращает все значения структур типа <typeparamref name="T"/> из файла находящегося в <paramref name="path"/>.</para>
        /// </summary>
        /// <typeparam name="T">
        /// <para>The structure type.</para>
        /// <para>Тип структуры.</para>
        /// </typeparam>
        /// <param name="path">
        /// <para>The path to the file, from which to read <typeparamref name="T"/> structure values array.</para>
        /// <para>Путь к файлу, из которого нужно прочитать массив значений структур типа <typeparamref name="T"/>.</para>
        /// </param>
        /// <returns>
        /// <para>The <typeparamref name="T"/> structure values array.</para>
        /// <para>Массив значений структур типа <typeparamref name="T"/>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T[] ReadAll<T>(string path)
            where T : struct
        {
            using var reader = File.OpenRead(path);
            return reader.ReadAll<T>();
        }

        /// <summary>
        /// <para>Reads and returns the first <typeparamref name="T"/> structure value from the file at the <paramref name="path"/>.</para>
        /// <para>Считывает и возвращает первое значение структуры типа <typeparamref name="T"/> из файла находящегося в <paramref name="path"/>.</para>
        /// </summary>
        /// <typeparam name="T">
        /// <para>The structure type.</para>
        /// <para>Тип структуры.</para>
        /// </typeparam>
        /// <param name="path">
        /// <para>The path to the file, from which to read the <typeparamref name="T"/> structure value.</para>
        /// <para>Путь к файлу, из которого нужно прочитать значение структуры типа <typeparamref name="T"/>.</para>
        /// </param>
        /// <returns>
        /// <para>The <typeparamref name="T"/> structure value if read from the file at the <paramref name="path"/> is successful; otherwise the default <typeparamref name="T"/> structure value.</para>
        /// <para>Значение структуры типа <typeparamref name="T"/> если чтение из файла находящегося в <paramref name="path"/> прошло успешно, иначе значение структуры типа <typeparamref name="T"/> по умолчанию.</para>
        /// </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
                ↪ <paramref name="path"/> if the file exists, not empty and its size is a multiple of
                ↪ the <typeparamref name="TStruct"/> structure size; otherwise <see
                ↪ langword="null"/>.</para>
81          /// <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>
82          /// </summary>
83          /// <typeparam name="TStruct">
84          /// <para>The structure type.</para>
85          /// <para>Тип структуры.</para>
86          /// </typeparam>
87          /// <param name="path">
88          /// <para>The path to the file to validate.</para>
89          /// <para>Путь к проверяемому файлу.</para>
90          /// </param>
91          /// <returns>
92          /// <para>A <see cref="FileStream"/> opened for reading in the case of successful check;
                ↪ otherwise <see langword="null"/>.</para>
93          /// <para><see cref="FileStream"/> открытый для чтения в случае успешной проверки, а
                ↪ иначе <see langword="null"/>.</para>
94          /// </returns>
95          /// <exception cref="InvalidOperationException">
96          /// <para>The size of the file at the <paramref name="path"/> is not a multiple of the
                ↪ required <paramref name="elementSize"/>.</para>
97          /// <para>Размер файла находящегося в <paramref name="path"/> не кратен требуемому
                ↪ <paramref name="elementSize"/>.</para>
98          /// </exception>
99          [MethodImpl(MethodImplOptions.AggressiveInlining)]
100         private static FileStream GetValidFileStreamOrDefault<TStruct>(string path) where
                ↪ TStruct : struct => GetValidFileStreamOrDefault(path, Structure<TStruct>.Size);
101
102         /// <summary>
103         /// <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>
104         /// <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>
105         /// </summary>
106         /// <param name="path">
107         /// <para>The path to the file to validate.</para>
108         /// <para>Путь к проверяемому файлу.</para>
109         /// </param>
110         /// <param name="elementSize">
111         /// <para>Required size of elements located in the file at the <paramref
                ↪ name="path"/>.</para>
112         /// <para>Требуемый размер элементов, находящихся в файле находящегося в <paramref
                ↪ name="path"/>.</para>
113         /// </param>
114         /// <returns>
115         /// <para>A <see cref="FileStream"/> opened for reading in the case of successful check;
                ↪ otherwise <see langword="null"/>.</para>
116         /// <para><see cref="FileStream"/> открытый для чтения в случае успешной проверки, а
                ↪ иначе <see langword="null"/>.</para>
117         /// </returns>
118         /// <exception cref="InvalidOperationException">
119         /// <para>The size of the file at the <paramref name="path"/> is not a multiple of the
                ↪ required <paramref name="elementSize"/>.</para>
120         /// <para>Размер файла находящегося в <paramref name="path"/> не кратен требуемому
                ↪ <paramref name="elementSize"/>.</para>
121         /// </exception>
122         [MethodImpl(MethodImplOptions.AggressiveInlining)]
123         private static FileStream GetValidFileStreamOrDefault(string path, int elementSize)
124         {
125             if (!File.Exists(path))
126             {
```

```
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
            ↪  соотвествии 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
            ↪  соотвествии 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>
```

```
        /// <para>The <typeparam name="T"> structure values array read from the <paramref
        ↪  name="stream"/>.</para>
        /// <para>Массив с значениями структур типа <typeparam name="T">, прочитанными из потока
        ↪  <paramref name="stream"/>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T[] ReadAll<T>(this Stream stream)
            where T : struct
        {
            var size = Structure<T>.Size;
            var buffer = new byte[size];
            var elementsLength = stream.Length / size;
            var elements = new T[elementsLength];
            for (var i = 0; i < elementsLength; i++)
            {
                stream.Read(buffer, 0, size);
                elements[i] = buffer.ToStructure<T>();
            }
            return elements;
        }
    }
}
```

1.5 ./csharp/Platform.IO.Tests/FileHelpersTests.cs

```
using System.IO;
using Xunit;

namespace Platform.IO.Tests
{
    public class FileHelpersTests
    {
        [Fact]
        public void WriteReadTest()
        {
            var temporaryFile = Path.GetTempFileName();
            var originalValue = 42UL;
            FileHelpers.WriteFirst(temporaryFile, originalValue);
            var readValue = FileHelpers.ReadFirstOrDefault<ulong>(temporaryFile);
            Assert.Equal(readValue, originalValue);
            File.Delete(temporaryFile);
        }
    }
}
```

Index