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
{
    public static class FileHelpers
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static char[] ReadAllChars(string path) => File.ReadAllText(path).ToCharArray();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T[] ReadAll<T>(string path)
            where T : struct
        {
            using var reader = File.OpenRead(path);
            return reader.ReadAll<T>();
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T ReadFirstOrDefault<T>(string path)
            where T : struct
        {
            using var fileStream = GetValidFileStreamOrDefault<T>(path);
            return fileStream?.ReadOrDefault<T>() ?? default;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private static FileStream GetValidFileStreamOrDefault<TStruct>(string path) where
        ↪  TStruct : struct => GetValidFileStreamOrDefault(path, Structure<TStruct>.Size);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private static FileStream GetValidFileStreamOrDefault(string path, int elementSize)
        {
            if (!File.Exists(path))
            {
                return null;
            }
            var fileSize = GetSize(path);
            if (fileSize % elementSize != 0)
            {
                throw new InvalidOperationException($"File is not aligned to elements with size
                ↪  {elementSize}.");
            }
            return fileSize > 0 ? File.OpenRead(path) : null;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T ReadLastOrDefault<T>(string path)
            where T : struct
        {
            var elementSize = Structure<T>.Size;
            using var reader = GetValidFileStreamOrDefault(path, elementSize);
            if (reader == null)
            {
                return default;
            }
            var totalElements = reader.Length / elementSize;
            reader.Position = (totalElements - 1) * elementSize; // Set to last element
            return reader.ReadOrDefault<T>();
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void WriteFirst<T>(string path, T value)
            where T : struct
        {
            using var writer = File.OpenWrite(path);
            writer.Position = 0;
            writer.Write(value);
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static FileStream Append(string path) => File.Open(path, FileMode.Append,
        ↪  FileAccess.Write);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static long GetSize(string path) => File.Exists(path) ? new FileInfo(path).Length
        ↪  : 0;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void SetSize(string path, long size)
        {
            using var fileStream = File.Open(path, FileMode.OpenOrCreate);
```

```
            if (fileStream.Length != size)
            {
                fileStream.SetLength(size);
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void DeleteAll(string directory) => DeleteAll(directory, "*");

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void DeleteAll(string directory, string searchPattern) =>
        ↪ DeleteAll(directory, searchPattern, SearchOption.TopDirectoryOnly);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void DeleteAll(string directory, string searchPattern, SearchOption
        ↪ searchOption)
        {
            foreach (var file in Directory.EnumerateFiles(directory, searchPattern,
            ↪ searchOption))
            {
                File.Delete(file);
            }
        }
    }
}
```

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

```
using System.IO;
using System.Runtime.CompilerServices;
using Platform.Unsafe;

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

namespace Platform.IO
{
    public static class StreamExtensions
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void Write<T>(this Stream stream, T value)
            where T : struct
        {
            var bytes = value.ToBytes();
            stream.Write(bytes, 0, bytes.Length);
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T ReadOrDefault<T>(this Stream stream)
            where T : struct
        {
            var size = Structure<T>.Size;
            var buffer = new byte[size];
            return stream.Read(buffer, 0, size) == size ? buffer.ToStructure<T>() : default;
        }

        [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