

LinksPlatform's Platform.IO Class Library

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

```
1  using System;
2  using System.Runtime.CompilerServices;
3  using System.Threading;
4  using Platform.Disposables;
5  using Platform.Threading;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.IO
10 {
11     public class ConsoleCancellation : DisposableBase
12     {
13         public CancellationTokenSource Source
14         {
15             [MethodImpl(MethodImplOptions.AggressiveInlining)]
16             get;
17         }
18
19         public CancellationToken Token
20         {
21             [MethodImpl(MethodImplOptions.AggressiveInlining)]
22             get;
23         }
24
25         public bool IsRequested
26         {
27             [MethodImpl(MethodImplOptions.AggressiveInlining)]
28             get => Source.IsCancellationRequested;
29         }
30
31         public bool NotRequested
32         {
33             [MethodImpl(MethodImplOptions.AggressiveInlining)]
34             get => !Source.IsCancellationRequested;
35         }
36
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public ConsoleCancellation()
39         {
40             Source = new CancellationTokenSource();
41             Token = Source.Token;
42             Console.CancelKeyPress += OnCancelKeyPress;
43         }
44
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public void ForceCancellation() => Source.Cancel();
47
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public void Wait()
50         {
51             while (NotRequested)
52             {
53                 ThreadHelpers.Sleep();
54             }
55         }
56
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         protected override void Dispose(bool manual, bool wasDisposed)
59         {
60             if (!wasDisposed)
61             {
62                 Console.CancelKeyPress -= OnCancelKeyPress;
63                 Source.DisposeIfPossible();
64             }
65         }
66
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         private void OnCancelKeyPress(object sender, ConsoleCancelEventArgs e)
69         {
70             e.Cancel = true;
71             if (NotRequested)
72             {
73                 Source.Cancel();
74             }
75         }
76     }
77 }
```

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

```
1 using System;
2 using System.Diagnostics;
3 using System.Runtime.CompilerServices;
4 using Platform.Collections;
5 using Platform.Collections.Arrays;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.IO
10 {
11     public static class ConsoleHelpers
12     {
13         /// <summary>
14         /// <para>Requests and expects a user to press any key in the console.</para>
15         /// <para>Запрашивает и ожидает нажатие любой клавиши пользователем в консоли.</para>
16         /// </summary>
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public static void PressAnyKeyToContinue()
19         {
20             Console.WriteLine("Press any key to continue.");
21             Console.ReadKey();
22         }
23
24         /// <summary>
25         /// <para>Gets the argument's value with the specified <paramref name="index" /> from
26         → the <paramref name="args"/> array and if it's absent requests a user to input it in
27         → the console.</para>
28         /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива
29         → <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у
30         → пользователя.</para>
31         /// </summary>
32         /// <param name="index">
33         /// <para>The ordinal number of the argument in the array.</para>
34         /// <para>Порядковый номер аргумента в массиве.</para>
35         /// </param>
36         /// <param name="args">
37         /// <para>The argument array passed to the application.</para>
38         /// <para>Массив аргументов переданных приложению.</para>
39         /// </param>
40         /// <returns>
41         /// <para>The value with the specified <paramref name="index"/> extracted from the
42         → <paramref name="args"/> array or entered by a user in the console.</para>
43         /// <para>Значение с указанным <paramref name="index"/>, извлечённое из массива
44         → <paramref name="args"/>, или введённое пользователем в консоли.</para>
45         /// </returns>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static string GetOrReadArgument(int index, params string[] args) =>
48         → GetOrReadArgument(index, $"{index + 1} argument", args);
49
50         /// <summary>
51         /// <para>Gets the argument's value with the specified <paramref name="index" /> from
52         → the <paramref name="args"/> array and if it's absent requests a user to input it in
53         → the console.</para>
54         /// <para>Получает значение аргумента с указанным <paramref name="index"/> из массива
55         → <paramref name="args"/>, а если оно отсутствует запрашивает его ввод в консоли у
56         → пользователя.</para>
57         /// </summary>
58         /// <param name="index">
59         /// <para>The ordinal number of the argument in the array.</para>
60         /// <para>Порядковый номер аргумента в массиве.</para>
61         /// </param>
62         /// <param name="readMessage">
63         /// <para>The message's text to a user describing which argument is being entered at the
64         → moment. If the <paramref name="args"/> array doesn't contain the element with the
65         → specified <paramref name="index"/>, then this message is used.</para>
66         /// <para>Текст сообщения пользователю описывающее какой аргумент вводится в данный
67         → момент. Это сообщение используется только если массив <paramref name="args"/> не
68         → содержит аргумента с указанным <paramref name="index"/>.</para>
69         /// </param>
70         /// <param name="args">
71         /// <para>The argument array passed to the application.</para>
72         /// <para>Массив аргументов переданных приложению.</para>
73         /// </param>
74         /// <returns>
75         /// <para>The value with the specified <paramref name="index"/> extracted from the
76         → <paramref name="args"/> array or entered by a user in the console.</para>
```

```

61  /// <para>Значение с указанным <paramref name="index"/>, извлечённое из массива
62  ↳ <paramref name="args"/>, или введённое пользователем в консоли.</para>
63  /// </returns>
64  [MethodImpl(MethodImplOptions.AggressiveInlining)]
65  public static string GetOrReadArgument(int index, string readMessage, params string[]
66  ↳ args)
67  {
68      if (!args.TryGetElement(index, out string result))
69      {
70          Console.Write($"{readMessage}: ");
71          result = Console.ReadLine();
72      }
73      if (string.IsNullOrEmpty(result))
74      {
75          return "";
76      }
77      else
78      {
79          return result.Trim().TrimSingle(' ').Trim();
80      }
81  }
82  /// <summary>
83  /// <para>Outputs the <paramref name="string" /> to the console.</para>
84  /// <para>Выводит <paramref name="string" /> в консоль.</para>
85  /// </summary>
86  /// <param name="string">
87  /// <para>The string to output to the console.</para>
88  /// <para>Строка выводимая в консоль.</para>
89  /// </param>
90  /// <remarks>
91  /// <para>The method is only executed if the application was compiled with the DEBUG
92  ↳ directive.</para>
93  /// <para>Метод выполняется только в том случае, если приложение было скомпилировано с
94  ↳ директивой DEBUG.</para>
95  /// </remarks>
96  [Conditional("DEBUG")]
97  public static void Debug(string @string) => Console.WriteLine(@string);
98  /// <summary>
99  /// <para>Writes text representations of objects of the specified <paramref
100  ↳ name="args"/> array, followed by the current line terminator, to the standard output
101  ↳ stream using the specified <paramref name="format"/>.</para>
102  /// <para>Записывает текстовые представления объектов заданного массива <paramref
103  ↳ name="args"/>, за которым следует текущий признак конца строки, в стандартный
104  ↳ выходной поток с использованием заданного <paramref name="format"/>.</para>
105  /// </summary>
106  /// <param name="format">
107  /// <para>The composite format string.</para>
108  /// <para>Строка составного формата.</para>
109  /// </param>
110  /// <param name="args">
111  /// <para>The object array to write to the standard output stream using <paramref
112  ↳ name="format" />.</para>
113  /// <para>Массив объектов для записи в стандартный выходной поток с использованием
114  ↳ <paramref name="format" />.</para>
115  /// </param>
116  /// <remarks>
117  /// <para>The method is only executed if the application was compiled with the DEBUG
118  ↳ directive.</para>
119  /// <para>Метод выполняется только в том случае, если приложение было скомпилировано с
120  ↳ директивой DEBUG.</para>
121  /// </remarks>
122  [Conditional("DEBUG")]
123  public static void Debug(string format, params object[] args) =>
124  ↳ Console.WriteLine(format, args);
125  }
126  }

```

1.3 ./csharp/Platform.IO/FileHelpers.cs

```

1  using System;
2  using System.IO;
3  using System.Runtime.CompilerServices;
4  using Platform.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7

```

```

8 namespace Platform.IO
9 {
10     public static class FileHelpers
11     {
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public static char[] ReadAllChars(string path) => File.ReadAllText(path).ToCharArray();
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public static T[] ReadAll<T>(string path)
17             where T : struct
18         {
19             using var reader = File.OpenRead(path);
20             return reader.ReadAll<T>();
21         }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static T ReadFirstOrDefault<T>(string path)
25             where T : struct
26         {
27             using var fileStream = GetValidFileStreamOrDefault<T>(path);
28             return fileStream?.ReadOrDefault<T>() ?? default;
29         }
30
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         private static FileStream GetValidFileStreamOrDefault<TStruct>(string path) where
33             ↪ TStruct : struct => GetValidFileStreamOrDefault(path, Structure<TStruct>.Size);
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         private static FileStream GetValidFileStreamOrDefault(string path, int elementSize)
37         {
38             if (!File.Exists(path))
39             {
40                 return null;
41             }
42             var fileSize = GetSize(path);
43             if (fileSize % elementSize != 0)
44             {
45                 throw new InvalidOperationException($"File is not aligned to elements with size
46                 ↪ {elementSize}.");
47             }
48             return fileSize > 0 ? File.OpenRead(path) : null;
49         }
50
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         public static T ReadLastOrDefault<T>(string path)
53             where T : struct
54         {
55             var elementSize = Structure<T>.Size;
56             using var reader = GetValidFileStreamOrDefault(path, elementSize);
57             if (reader == null)
58             {
59                 return default;
60             }
61             var totalElements = reader.Length / elementSize;
62             reader.Position = (totalElements - 1) * elementSize; // Set to last element
63             return reader.ReadOrDefault<T>();
64         }
65
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         public static void WriteFirst<T>(string path, T value)
68             where T : struct
69         {
70             using var writer = File.OpenWrite(path);
71             writer.Position = 0;
72             writer.Write(value);
73         }
74
75         [MethodImpl(MethodImplOptions.AggressiveInlining)]
76         public static FileStream Append(string path) => File.Open(path, FileMode.Append,
77             ↪ FileAccess.Write);
78
79         [MethodImpl(MethodImplOptions.AggressiveInlining)]
80         public static long GetSize(string path) => File.Exists(path) ? new FileInfo(path).Length
81             ↪ : 0;
82
83         [MethodImpl(MethodImplOptions.AggressiveInlining)]
84         public static void SetSize(string path, long size)
85         {
86             using var fileStream = File.Open(path, FileMode.OpenOrCreate);
87         }
88     }
89 }

```

```

83         if (fileStream.Length != size)
84         {
85             fileStream.SetLength(size);
86         }
87     }
88
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public static void DeleteAll(string directory) => DeleteAll(directory, "*");
91
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public static void DeleteAll(string directory, string searchPattern) =>
94         ↪ DeleteAll(directory, searchPattern, SearchOption.TopDirectoryOnly);
95
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public static void DeleteAll(string directory, string searchPattern, SearchOption
98         ↪ searchOption)
99     {
100         foreach (var file in Directory.EnumerateFiles(directory, searchPattern,
101             ↪ searchOption))
102         {
103             File.Delete(file);
104         }
105     }
106 }

```

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      public static class StreamExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static void Write<T>(this Stream stream, T value)
13             where T : struct
14         {
15             var bytes = value.ToBytes();
16             stream.Write(bytes, 0, bytes.Length);
17         }
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public static T ReadOrDefault<T>(this Stream stream)
21             where T : struct
22         {
23             var size = Structure<T>.Size;
24             var buffer = new byte[size];
25             return stream.Read(buffer, 0, size) == size ? buffer.ToStructure<T>() : default;
26         }
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public static T[] ReadAll<T>(this Stream stream)
30             where T : struct
31         {
32             var size = Structure<T>.Size;
33             var buffer = new byte[size];
34             var elementsLength = stream.Length / size;
35             var elements = new T[elementsLength];
36             for (var i = 0; i < elementsLength; i++)
37             {
38                 stream.Read(buffer, 0, size);
39                 elements[i] = buffer.ToStructure<T>();
40             }
41             return elements;
42         }
43     }
44 }

```

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

```

1  using System.IO;
2  using Xunit;
3
4  namespace Platform.IO.Tests
5  {
6      public class FileHelpersTests

```

```
7 {
8     [Fact]
9     public void WriteReadTest()
10    {
11        var temporaryFile = Path.GetTempFileName();
12        var originalValue = 42UL;
13        FileHelpers.WriteFirst(temporaryFile, originalValue);
14        var readValue = FileHelpers.ReadFirstOrDefault<ulong>(temporaryFile);
15        Assert.Equal(readValue, originalValue);
16        File.Delete(temporaryFile);
17    }
18 }
19 }
```

Index

- ./csharp/Platform.IO.Tests/FileHelpersTests.cs, 5
- ./csharp/Platform.IO/ConsoleCancellation.cs, 1
- ./csharp/Platform.IO/ConsoleHelpers.cs, 1
- ./csharp/Platform.IO/FileHelpers.cs, 3
- ./csharp/Platform.IO/StreamExtensions.cs, 5