

LinksPlatform's Platform.IO Class Library

./ConsoleCancellation.cs

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

./ConsoleHelpers.cs

```
1 using System;
2 using System.Diagnostics;
3 using Platform.Collections;
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 ConsoleHelpers
10     {
11         public static void PressAnyKeyToContinue()
12         {
13             Console.WriteLine("Press any key to continue.");
14             Console.ReadKey();
15         }
16
17         public static string GetOrReadArgument(int index, params string[] args) =>
18             ↪ GetOrReadArgument(index, $"{index + 1} argument", args);
19
20         public static string GetOrReadArgument(int index, string readMessage, params string[]
21             ↪ args)
22         {
23             if (index < 0 || index > args.Length)
24             {
25                 throw new ArgumentOutOfRangeException(index, readMessage, "Invalid argument index");
26             }
27             return args[index];
28         }
29     }
30 }
```

```

20     {
21         string result;
22         if (args != null && args.Length > index)
23         {
24             result = args[index];
25         }
26         else
27         {
28             Console.Write($"{readMessage}: ");
29             result = Console.ReadLine();
30         }
31         result = (result ?? "").Trim().TrimSingle(' ').Trim();
32         return result;
33     }
34
35     [Conditional("DEBUG")]
36     public static void Debug(string format, params object[] args) =>
37         ↪ Console.WriteLine(format, args);
38 }

```

./FileHelpers.cs

```

1  using System;
2  using System.IO;
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 FileHelpers
10     {
11         public static char[] ReadAllChars(string path) => File.ReadAllText(path).ToCharArray();
12
13         public static T[] ReadAll<T>(string path)
14             where T : struct
15         {
16             using (var reader = File.OpenRead(path))
17             {
18                 return reader.ReadAll<T>();
19             }
20         }
21
22         public static T ReadFirstOrDefault<T>(string path)
23             where T : struct
24         {
25             using (var fileStream = GetValidFileStreamOrDefault<T>(path))
26             {
27                 return fileStream?.ReadOrDefault<T>() ?? default;
28             }
29         }
30
31         private static FileStream GetValidFileStreamOrDefault<TStruct>(string path) where
32             ↪ TStruct : struct => GetValidFileStreamOrDefault(path, Structure<TStruct>.Size);
33
34         private static FileStream GetValidFileStreamOrDefault(string path, int elementSize)
35         {
36             if (!File.Exists(path))
37             {
38                 return null;
39             }
40             var fileSize = GetSize(path);
41             if (fileSize % elementSize != 0)
42             {
43                 throw new InvalidOperationException($"{File is not aligned to elements with size
44                     ↪ {elementSize}.");
45             }
46             return fileSize > 0 ? File.OpenRead(path) : null;
47         }
48
49         public static T ReadLastOrDefault<T>(string path)
50             where T : struct
51         {
52             var elementSize = Structure<T>.Size;
53             using (var reader = GetValidFileStreamOrDefault(path, elementSize))
54             {
55                 if (reader == null)
56                 {
57                     return default;
58                 }
59             }
60         }
61     }
62 }

```

```

56     }
57     var totalElements = reader.Length / elementSize;
58     reader.Position = (totalElements - 1) * elementSize; // Set to last element
59     return reader.ReadOrDefault<T>();
60 }
61 }
62
63 public static void WriteFirst<T>(string path, T value)
64     where T : struct
65 {
66     using (var writer = File.OpenWrite(path))
67     {
68         writer.Position = 0;
69         writer.Write(value);
70     }
71 }
72
73 public static FileStream Append(string path) => File.Open(path, FileMode.Append,
    ↪ FileAccess.Write);
74
75 public static long GetSize(string path) => File.Exists(path) ? new FileInfo(path).Length
    ↪ : 0;
76
77 public static void SetSize(string path, long size)
78 {
79     using (var fileStream = File.Open(path, FileMode.OpenOrCreate))
80     {
81         if (fileStream.Length != size)
82         {
83             fileStream.SetLength(size);
84         }
85     }
86 }
87 }
88 }

```

./StreamExtensions.cs

```

1  using System.IO;
2  using Platform.Unsafe;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.IO
7  {
8      public static class StreamExtensions
9      {
10         public static void Write<T>(this Stream stream, T value)
11             where T : struct
12         {
13             var bytes = value.ToBytes();
14             stream.Write(bytes, 0, bytes.Length);
15         }
16
17         public static T ReadOrDefault<T>(this Stream stream)
18             where T : struct
19         {
20             var size = Structure<T>.Size;
21             var buffer = new byte[size];
22             return stream.Read(buffer, 0, size) == size ? buffer.ToStructure<T>() : default;
23         }
24
25         public static T[] ReadAll<T>(this Stream stream)
26             where T : struct
27         {
28             var size = Structure<T>.Size;
29             var buffer = new byte[size];
30             var elementsLength = stream.Length / size;
31             var elements = new T[elementsLength];
32             for (var i = 0; i < elementsLength; i++)
33             {
34                 stream.Read(buffer, 0, size);
35                 elements[i] = buffer.ToStructure<T>();
36             }
37             return elements;
38         }
39     }
40 }

```

Index

./ConsoleCancellation.cs, 1
./ConsoleHelpers.cs, 1
./FileHelpers.cs, 2
./StreamExtensions.cs, 3