```csharp
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

```csharp
1  using System;
2  using System.Diagnostics;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections;
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 ConsoleHelpers
11     {
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public static void PressAnyKeyToContinue()
14         {
15             Console.WriteLine("Press any key to continue.");
16             Console.ReadKey();
17         }
18
19         [MethodImpl(MethodImplOptions.AggressiveInlining)]
20         public static string GetOrReadArgument(int index, params string[] args) =>
21         ↪   GetOrReadArgument(index, $"{index + 1} argument", args);
22
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         public static string GetOrReadArgument(int index, string readMessage, params string[]
24         ↪   args)
24         {
25             string result;
26             if (args != null && args.Length > index)
27             {
28                 result = args[index];
29             }
30             else
31             {
32                 Console.Write($"{readMessage}: ");
33                 result = Console.ReadLine();
34             }
35             result = (result ?? "").Trim().TrimSingle('"').Trim();
36             return result;
37         }
38
39         [Conditional("DEBUG")]
40         public static void Debug(string format, params object[] args) =>
41         ↪   Console.WriteLine(format, args);
41     }
42  }
```

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

```csharp
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)]
```

```csharp
    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);
        }
    }
```

```
103          }
104      }
```

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