

## LinksPlatform's Platform.Hardware.Cpu Class Library

### 1.1 ./csharp/Platform.Hardware.Cpu/CacheLine.cs

```
1 using System;
2 using System.Runtime.InteropServices;
3
4 namespace Platform.Hardware.Cpu
5 {
6     /// <summary>
7     /// <para>Contains constants related to CPUs cache line.</para>
8     /// <para>Содержит константы, относящиеся к строке кэша ЦП.</para>
9     /// </summary>
10    public static class CacheLine
11    {
12        /// <summary>
13        /// <para>Gets the size of CPUs cache line in bytes.</para>
14        /// <para>Получает размер строки кэша ЦП в байтах.</para>
15        /// </summary>
16        public static readonly int Size = GetSize();
17
18        private static int GetSize()
19        {
20            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
21            {
22                return Windows.GetSize();
23            }
24            if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
25            {
26                return Linux.GetSize();
27            }
28            if (RuntimeInformation.IsOSPlatform(OSPlatform.OSX))
29            {
30                return OSX.GetSize();
31            }
32            throw new NotSupportedException("Unrecognized OS platform.");
33        }
34    }
35 }
```

### 1.2 ./csharp/Platform.Hardware.Cpu/Linux.cs

```
1 using System;
2 using System.Runtime.InteropServices;
3
4 #pragma warning disable IDE1006 // Naming Styles
5
6 namespace Platform.Hardware.Cpu
7 {
8     internal static class Linux
9     {
10        public static int GetSize() => (int)sysconf(_SC_LEVEL1_DCACHE_LINESIZE);
11
12        [DllImport("libc")]
13        private static extern Int64 sysconf(Int32 name);
14
15        private const Int32 _SC_LEVEL1_DCACHE_LINESIZE = 190;
16    }
17 }
```

### 1.3 ./csharp/Platform.Hardware.Cpu/OSX.cs

```
1 using System;
2 using System.Runtime.InteropServices;
3
4 #pragma warning disable IDE1006 // Naming Styles
5
6 namespace Platform.Hardware.Cpu
7 {
8     internal static class OSX
9     {
10        public static int GetSize()
11        {
12            var sizeOfLineSize = (IntPtr)IntPtr.Size;
13            sysctlbyname("hw.cachelinesize", out IntPtr lineSize, ref sizeOfLineSize,
14                ↪ IntPtr.Zero, IntPtr.Zero);
15            return lineSize.ToInt32();
16        }
17
18        [DllImport("libc")]
19        private static extern Int32 sysctlbyname(string name, out IntPtr oldp, ref IntPtr
20            ↪ oldlenp, IntPtr newp, IntPtr newlen);
21    }
22 }
```

```
19     }
20 }
```

#### 1.4 ./csharp/Platform.Hardware.Cpu/Windows.cs

```
1 using System;
2 using System.Linq;
3 using System.Runtime.InteropServices;
4
5 #pragma warning disable 0649
6 #pragma warning disable IDE0044 // Add readonly modifier
7
8 namespace Platform.Hardware.Cpu
9 {
10     internal static class Windows
11     {
12         public static int GetSize()
13         {
14             var info = ManagedGetLogicalProcessorInformation();
15             if (info == null)
16             {
17                 throw new InvalidOperationException("Could not retrieve the cache line size.");
18             }
19             return info.First(x => x.Relationship == LOGICAL_PROCESSOR_RELATIONSHIP.RelationCache)
20                 ↪ e).ProcessorInformation.Cache.LineSize;
21         }
22
23         // http://stackoverflow.com/a/6972620/232574
24         [StructLayout(LayoutKind.Sequential)]
25         struct PROCESSORCORE
26         {
27             public byte Flags;
28         }
29
30         [StructLayout(LayoutKind.Sequential)]
31         struct NUMANODE
32         {
33             public uint NodeNumber;
34         }
35
36         enum PROCESSOR_CACHE_TYPE
37         {
38             CacheUnified,
39             CacheInstruction,
40             CacheData,
41             CacheTrace
42         }
43
44         [StructLayout(LayoutKind.Sequential)]
45         struct CACHE_DESCRIPTOR
46         {
47             public Byte Level;
48             public Byte Associativity;
49             public UInt16 LineSize;
50             public UInt32 Size;
51             public PROCESSOR_CACHE_TYPE Type;
52         }
53
54         [StructLayout(LayoutKind.Explicit)]
55         struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION
56         {
57             [FieldOffset(0)]
58             public PROCESSORCORE ProcessorCore;
59             [FieldOffset(0)]
60             public NUMANODE NumaNode;
61             [FieldOffset(0)]
62             public CACHE_DESCRIPTOR Cache;
63             [FieldOffset(0)]
64             private UInt64 Reserved1;
65             [FieldOffset(8)]
66             private UInt64 Reserved2;
67         }
68
69         enum LOGICAL_PROCESSOR_RELATIONSHIP
70         {
71             RelationProcessorCore,
72             RelationNumaNode,
73             RelationCache,
74             RelationProcessorPackage,
75             RelationGroup,
76             RelationAll = 0xffff
77         }
78     }
79 }
```

```

77     private struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION
78     {
79     {
80         public UIntPtr ProcessorMask;
81         public LOGICAL_PROCESSOR_RELATIONSHIP Relationship;
82         public SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION ProcessorInformation;
83     }
84
85     [DllImport(@"kernel32.dll", SetLastError = true)]
86     private static extern bool GetLogicalProcessorInformation(IntPtr Buffer, ref UInt32
        ↳ ReturnLength);
87
88     private const int ERROR_INSUFFICIENT_BUFFER = 122;
89
90     private static SYSTEM_LOGICAL_PROCESSOR_INFORMATION[]
        ↳ ManagedGetLogicalProcessorInformation()
91     {
92         var ReturnLength = 0u;
93         GetLogicalProcessorInformation(IntPtr.Zero, ref ReturnLength);
94         if (Marshal.GetLastWin32Error() != ERROR_INSUFFICIENT_BUFFER)
95         {
96             return null;
97         }
98         var pointer = Marshal.AllocHGlobal((int)ReturnLength);
99         try
100         {
101             if (GetLogicalProcessorInformation(pointer, ref ReturnLength))
102             {
103                 var size = Marshal.SizeOf<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>();
104                 var length = (int)ReturnLength / size;
105                 var buffer = new SYSTEM_LOGICAL_PROCESSOR_INFORMATION[length];
106                 var itemPointer = pointer;
107                 for (int i = 0; i < length; i++)
108                 {
109                     buffer[i] = Marshal.PtrToStructure<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>(
        ↳ itemPointer);
110                     itemPointer += size;
111                 }
112                 return buffer;
113             }
114         }
115         finally
116         {
117             Marshal.FreeHGlobal(pointer);
118         }
119         return null;
120     }
121 }
122 }

```

## 1.5 ./csharp/Platform.Hardware.Cpu.Tests/CacheLineTests.cs

```

1  using Xunit;
2
3  namespace Platform.Hardware.Cpu.Tests
4  {
5      public static class Tests
6      {
7          [Fact]
8          public static void Test() => Assert.NotEqual(0, CacheLine.Size);
9      }
10 }

```

## Index

- ./csharp/Platform.Hardware.Cpu.Tests/CacheLineTests.cs, 3
- ./csharp/Platform.Hardware.Cpu/CacheLine.cs, 1
- ./csharp/Platform.Hardware.Cpu/Linux.cs, 1
- ./csharp/Platform.Hardware.Cpu/OSX.cs, 1
- ./csharp/Platform.Hardware.Cpu/Windows.cs, 2