

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.RelationCach
                ↪ e).ProcessorInformation.Cache.LineSize;
20         }
21
22         // http://stackoverflow.com/a/6972620/232574
23         [StructLayout(LayoutKind.Sequential)]
24         struct PROCESSORCORE
25         {
26             public byte Flags;
27         }
28
29         [StructLayout(LayoutKind.Sequential)]
30         struct NUMANODE
31         {
32             public uint NodeNumber;
33         }
34
35         enum PROCESSOR_CACHE_TYPE
36         {
37             CacheUnified,
38             CacheInstruction,
39             CacheData,
40             CacheTrace
41         }
42
43         [StructLayout(LayoutKind.Sequential)]
44         struct CACHE_DESCRIPTOR
45         {
46             public byte Level;
47             public byte Associativity;
48             public UInt16 LineSize;
49             public UInt32 Size;
50             public PROCESSOR_CACHE_TYPE Type;
51         }
52
53         [StructLayout(LayoutKind.Explicit)]
54         struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION
55         {
56             [FieldOffset(0)]
57             public PROCESSORCORE ProcessorCore;
58             [FieldOffset(0)]
59             public NUMANODE NumaNode;
60             [FieldOffset(0)]
61             public CACHE_DESCRIPTOR Cache;
62             [FieldOffset(0)]
63             private UInt64 Reserved1;
64             [FieldOffset(8)]
65             private UInt64 Reserved2;
66         }
67
68         enum LOGICAL_PROCESSOR_RELATIONSHIP
69         {
70             RelationProcessorCore,
71             RelationNumaNode,
72             RelationCache,
73             RelationProcessorPackage,
74             RelationGroup,
75             RelationAll = 0xffff
76         }

```

```

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