

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        private static int GetSize()
18        {
19            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
20            {
21                return Windows.GetSize();
22            }
23            if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
24            {
25                return Linux.GetSize();
26            }
27            if (RuntimeInformation.IsOSPlatform(OSPlatform.OSX))
28            {
29                return OSX.GetSize();
30            }
31            throw new NotSupportedException("Unrecognized OS platform.");
32        }
33    }
34 }
```

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     /// <summary>
9     /// <para>
10    /// Represents the linux.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    internal static class Linux
15    {
16        /// <summary>
17        /// <para>
18        /// Gets the size.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <returns>
23        /// <para>The int</para>
24        /// <para></para>
25        /// </returns>
26        public static int GetSize() => (int)sysconf(_SC_LEVEL1_DCACHE_LINESIZE);
27
28        /// <summary>
29        /// <para>
30        /// Sysconfs the name.
31        /// </para>
32        /// <para></para>
33        /// </summary>
34        /// <param name="name">
35        /// <para>The name.</para>
36        /// <para></para>
37        /// </param>
38        /// <returns>
39        /// <para>The int 64</para>
40        /// <para></para>
```

```

41     /// </returns>
42     [DllImport("libc")]
43     private static extern Int64 sysconf(Int32 name);
44     private const Int32 _SC_LEVEL1_DCACHE_LINESIZE = 190;
45 }
46 }

```

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     /// <summary>
9     /// <para>
10    /// Represents the osx.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    internal static class OSX
15    {
16        /// <summary>
17        /// <para>
18        /// Gets the size.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <returns>
23        /// <para>The int</para>
24        /// <para></para>
25        /// </returns>
26        public static int GetSize()
27        {
28            var sizeOfLineSize = (IntPtr)IntPtr.Size;
29            sysctlbyname("hw.cachelinesize", out IntPtr lineSize, ref sizeOfLineSize,
30                ↪ IntPtr.Zero, IntPtr.Zero);
31            return lineSize.ToInt32();
32        }
33
34        /// <summary>
35        /// <para>
36        /// Sysctlbyname the name.
37        /// </para>
38        /// <para></para>
39        /// </summary>
40        /// <param name="name">
41        /// <para>The name.</para>
42        /// <para></para>
43        /// </param>
44        /// <param name="oldp">
45        /// <para>The oldp.</para>
46        /// <para></para>
47        /// </param>
48        /// <param name="oldlenp">
49        /// <para>The oldlenp.</para>
50        /// <para></para>
51        /// </param>
52        /// <param name="newp">
53        /// <para>The newp.</para>
54        /// <para></para>
55        /// </param>
56        /// <param name="newlen">
57        /// <para>The newlen.</para>
58        /// <para></para>
59        /// </param>
60        /// <returns>
61        /// <para>The int 32</para>
62        /// <para></para>
63        /// </returns>
64        [DllImport("libc")]
65        private static extern Int32 sysctlbyname(string name, out IntPtr oldp, ref IntPtr
66            ↪ oldlenp, IntPtr newp, IntPtr newlen);

```

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     /// <summary>
11     /// <para>
12     /// Represents the windows.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     internal static class Windows
17     {
18         /// <summary>
19         /// <para>
20         /// Gets the size.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <exception cref="InvalidOperationException">
25         /// <para>Could not retrieve the cache line size.</para>
26         /// <para></para>
27         /// </exception>
28         /// <returns>
29         /// <para>The int</para>
30         /// <para></para>
31         /// </returns>
32         public static int GetSize()
33         {
34             var info = ManagedGetLogicalProcessorInformation();
35             if (info == null)
36             {
37                 throw new InvalidOperationException("Could not retrieve the cache line size.");
38             }
39             return info.First(x => x.Relationship == LOGICAL_PROCESSOR_RELATIONSHIP.RelationCach
40                 ↪ e).ProcessorInformation.Cache.LineSize;
41         }
42
43         // http://stackoverflow.com/a/6972620/232574
44         /// <summary>
45         /// <para>
46         /// The processorcore.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         [StructLayout(LayoutKind.Sequential)]
51         struct PROCESSORCORE
52         {
53             /// <summary>
54             /// <para>
55             /// The flags.
56             /// </para>
57             /// <para></para>
58             /// </summary>
59             public byte Flags;
60
61             /// <summary>
62             /// <para>
63             /// The numanode.
64             /// </para>
65             /// <para></para>
66             /// </summary>
67             [StructLayout(LayoutKind.Sequential)]
68             struct NUMANODE
69             {
70                 /// <summary>
71                 /// <para>
72                 /// The node number.
73                 /// </para>
74                 /// <para></para>
75                 /// </summary>
76                 public uint NodeNumber;
77             }

```

```

78
79     /// <summary>
80     /// <para>
81     /// The processor cache type enum.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     enum PROCESSOR_CACHE_TYPE
86     {
87         /// <summary>
88         /// <para>
89         /// The cache unified processor cache type.
90         /// </para>
91         /// <para></para>
92         /// </summary>
93         CacheUnified,
94         /// <summary>
95         /// <para>
96         /// The cache instruction processor cache type.
97         /// </para>
98         /// <para></para>
99         /// </summary>
100        CacheInstruction,
101        /// <summary>
102        /// <para>
103        /// The cache data processor cache type.
104        /// </para>
105        /// <para></para>
106        /// </summary>
107        CacheData,
108        /// <summary>
109        /// <para>
110        /// The cache trace processor cache type.
111        /// </para>
112        /// <para></para>
113        /// </summary>
114        CacheTrace
115    }
116
117     /// <summary>
118     /// <para>
119     /// The cache descriptor.
120     /// </para>
121     /// <para></para>
122     /// </summary>
123     [StructLayout(LayoutKind.Sequential)]
124     struct CACHE_DESCRIPTOR
125     {
126         /// <summary>
127         /// <para>
128         /// The level.
129         /// </para>
130         /// <para></para>
131         /// </summary>
132         public Byte Level;
133         /// <summary>
134         /// <para>
135         /// The associativity.
136         /// </para>
137         /// <para></para>
138         /// </summary>
139         public Byte Associativity;
140         /// <summary>
141         /// <para>
142         /// The line size.
143         /// </para>
144         /// <para></para>
145         /// </summary>
146         public UInt16 LineSize;
147         /// <summary>
148         /// <para>
149         /// The size.
150         /// </para>
151         /// <para></para>
152         /// </summary>
153         public UInt32 Size;
154         /// <summary>
155         /// <para>

```

```

156     /// The type.
157     /// </para>
158     /// <para></para>
159     /// </summary>
160     public PROCESSOR_CACHE_TYPE Type;
161 }
162
163 /// <summary>
164 /// <para>
165 /// The system logical processor information union.
166 /// </para>
167 /// <para></para>
168 /// </summary>
169 [StructLayout(LayoutKind.Explicit)]
170 struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION
171 {
172     /// <summary>
173     /// <para>
174     /// The processor core.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     [FieldOffset(0)]
179     public PROCESSORCORE ProcessorCore;
180     /// <summary>
181     /// <para>
182     /// The numa node.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     [FieldOffset(0)]
187     public NUMANODE NumaNode;
188     /// <summary>
189     /// <para>
190     /// The cache.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     [FieldOffset(0)]
195     public CACHE_DESCRIPTOR Cache;
196     /// <summary>
197     /// <para>
198     /// The reserved.
199     /// </para>
200     /// <para></para>
201     /// </summary>
202     [FieldOffset(0)]
203     private UInt64 Reserved1;
204     /// <summary>
205     /// <para>
206     /// The reserved.
207     /// </para>
208     /// <para></para>
209     /// </summary>
210     [FieldOffset(8)]
211     private UInt64 Reserved2;
212 }
213
214 /// <summary>
215 /// <para>
216 /// The logical processor relationship enum.
217 /// </para>
218 /// <para></para>
219 /// </summary>
220 enum LOGICAL_PROCESSOR_RELATIONSHIP
221 {
222     /// <summary>
223     /// <para>
224     /// The relation processor core logical processor relationship.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     RelationProcessorCore,
229     /// <summary>
230     /// <para>
231     /// The relation numa node logical processor relationship.
232     /// </para>
233     /// <para></para>

```

```

234     /// </summary>
235     RelationNumaNode,
236     /// <summary>
237     /// <para>
238     /// The relation cache logical processor relationship.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     RelationCache,
243     /// <summary>
244     /// <para>
245     /// The relation processor package logical processor relationship.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     RelationProcessorPackage,
250     /// <summary>
251     /// <para>
252     /// The relation group logical processor relationship.
253     /// </para>
254     /// <para></para>
255     /// </summary>
256     RelationGroup,
257     /// <summary>
258     /// <para>
259     /// The relation all logical processor relationship.
260     /// </para>
261     /// <para></para>
262     /// </summary>
263     RelationAll = 0xffff
264 }
265 private struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION
266 {
267     /// <summary>
268     /// <para>
269     /// The processor mask.
270     /// </para>
271     /// <para></para>
272     /// </summary>
273     public UIntPtr ProcessorMask;
274     /// <summary>
275     /// <para>
276     /// The relationship.
277     /// </para>
278     /// <para></para>
279     /// </summary>
280     public LOGICAL_PROCESSOR_RELATIONSHIP Relationship;
281     /// <summary>
282     /// <para>
283     /// The processor information.
284     /// </para>
285     /// <para></para>
286     /// </summary>
287     public SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION ProcessorInformation;
288 }
289
290 /// <summary>
291 /// <para>
292 /// Determines whether get logical processor information.
293 /// </para>
294 /// <para></para>
295 /// </summary>
296 /// <param name="Buffer">
297 /// <para>The buffer.</para>
298 /// <para></para>
299 /// </param>
300 /// <param name="ReturnLength">
301 /// <para>The return length.</para>
302 /// <para></para>
303 /// </param>
304 /// <returns>
305 /// <para>The bool</para>
306 /// <para></para>
307 /// </returns>
308 [DllImport(@"kernel32.dll", SetLastError = true)]
309 private static extern bool GetLogicalProcessorInformation(IntPtr Buffer, ref UInt32
↪ ReturnLength);
310 private const int ERROR_INSUFFICIENT_BUFFER = 122;

```

```

311 private static SYSTEM_LOGICAL_PROCESSOR_INFORMATION[]
    ↳ ManagedGetLogicalProcessorInformation()
312 {
313     var ReturnLength = 0u;
314     GetLogicalProcessorInformation(IntPtr.Zero, ref ReturnLength);
315     if (Marshal.GetLastWin32Error() != ERROR_INSUFFICIENT_BUFFER)
316     {
317         return null;
318     }
319     var pointer = Marshal.AllocHGlobal((int)ReturnLength);
320     try
321     {
322         if (GetLogicalProcessorInformation(pointer, ref ReturnLength))
323         {
324             var size = Marshal.SizeOf<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>();
325             var length = (int)ReturnLength / size;
326             var buffer = new SYSTEM_LOGICAL_PROCESSOR_INFORMATION[length];
327             var itemPointer = pointer;
328             for (int i = 0; i < length; i++)
329             {
330                 buffer[i] = Marshal.PtrToStructure<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>
    ↳ (itemPointer);
331                 itemPointer += size;
332             }
333             return buffer;
334         }
335     }
336     finally
337     {
338         Marshal.FreeHGlobal(pointer);
339     }
340     return null;
341 }
342 }
343 }

```

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, 7
- ./csharp/Platform.Hardware.Cpu/CacheLine.cs, 1
- ./csharp/Platform.Hardware.Cpu/Linux.cs, 1
- ./csharp/Platform.Hardware.Cpu/OSX.cs, 2
- ./csharp/Platform.Hardware.Cpu/Windows.cs, 2