

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        /// <summary>
19        /// <para>
20        /// Gets the size.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <exception cref="NotSupportedException">
25        /// <para>Unrecognized OS platform.</para>
26        /// <para></para>
27        /// </exception>
28        /// <returns>
29        /// <para>The int</para>
30        /// <para></para>
31        /// </returns>
32        private static int GetSize()
33        {
34            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
35            {
36                return Windows.GetSize();
37            }
38            if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
39            {
40                return Linux.GetSize();
41            }
42            if (RuntimeInformation.IsOSPlatform(OSPlatform.OSX))
43            {
44                return OSX.GetSize();
45            }
46            throw new NotSupportedException("Unrecognized OS platform.");
47        }
48    }
49 }
```

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
45     /// <summary>
46     /// <para>
47     /// The sc level1 dcache linesize.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     private const Int32 _SC_LEVEL1_DCACHE_LINESIZE = 190;
52 }
53 }

```

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">

```

```

48     /// <para>The oldlenp.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="newp">
52     /// <para>The newp.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="newlen">
56     /// <para>The newlen.</para>
57     /// <para></para>
58     /// </param>
59     /// <returns>
60     /// <para>The int 32</para>
61     /// <para></para>
62     /// </returns>
63     [DllImport("libc")]
64     private static extern Int32 sysctlbyname(string name, out IntPtr oldp, ref IntPtr
        ↪ oldlenp, IntPtr newp, IntPtr newlen);
65 }
66 }

```

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
        ↪ e).ProcessorInformation.Cache.LineSize;
40         }
41
42         // http://stackoverflow.com/a/6972620/232574
43         /// <summary>
44         /// <para>
45         /// The processorcore.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         [StructLayout(LayoutKind.Sequential)]
50         struct PROCESSORCORE
51         {
52             /// <summary>
53             /// <para>
54             /// The flags.
55             /// </para>

```

```

56     /// <para></para>
57     /// </summary>
58     public byte Flags;
59 }
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
266 /// <summary>
267 /// <para>
268 /// The system logical processor information.
269 /// </para>
270 /// <para></para>
271 /// </summary>
272 private struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION
273 {
274     /// <summary>
275     /// <para>
276     /// The processor mask.
277     /// </para>
278     /// <para></para>
279     /// </summary>
280     public UIntPtr ProcessorMask;
281     /// <summary>
282     /// <para>
283     /// The relationship.
284     /// </para>
285     /// <para></para>
286     /// </summary>
287     public LOGICAL_PROCESSOR_RELATIONSHIP Relationship;
288     /// <summary>
289     /// <para>
290     /// The processor information.
291     /// </para>

```

```

292     /// <para></para>
293     /// </summary>
294     public SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION ProcessorInformation;
295 }
296
297 /// <summary>
298 /// <para>
299 /// Determines whether get logical processor information.
300 /// </para>
301 /// <para></para>
302 /// </summary>
303 /// <param name="Buffer">
304 /// <para>The buffer.</para>
305 /// <para></para>
306 /// </param>
307 /// <param name="ReturnLength">
308 /// <para>The return length.</para>
309 /// <para></para>
310 /// </param>
311 /// <returns>
312 /// <para>The bool</para>
313 /// <para></para>
314 /// </returns>
315 [DllImport(@"kernel32.dll", SetLastError = true)]
316 private static extern bool GetLogicalProcessorInformation(IntPtr Buffer, ref UInt32
    ↳ ReturnLength);
317
318 /// <summary>
319 /// <para>
320 /// The error insufficient buffer.
321 /// </para>
322 /// <para></para>
323 /// </summary>
324 private const int ERROR_INSUFFICIENT_BUFFER = 122;
325
326 /// <summary>
327 /// <para>
328 /// Manages the get logical processor information.
329 /// </para>
330 /// <para></para>
331 /// </summary>
332 /// <returns>
333 /// <para>The system logical processor information array</para>
334 /// <para></para>
335 /// </returns>
336 private static SYSTEM_LOGICAL_PROCESSOR_INFORMATION[]
    ↳ ManagedGetLogicalProcessorInformation()
337 {
338     var ReturnLength = 0u;
339     GetLogicalProcessorInformation(IntPtr.Zero, ref ReturnLength);
340     if (Marshal.GetLastWin32Error() != ERROR_INSUFFICIENT_BUFFER)
341     {
342         return null;
343     }
344     var pointer = Marshal.AllocHGlobal((int)ReturnLength);
345     try
346     {
347         if (GetLogicalProcessorInformation(pointer, ref ReturnLength))
348         {
349             var size = Marshal.SizeOf<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>();
350             var length = (int)ReturnLength / size;
351             var buffer = new SYSTEM_LOGICAL_PROCESSOR_INFORMATION[length];
352             var itemPointer = pointer;
353             for (int i = 0; i < length; i++)
354             {
355                 buffer[i] = Marshal.PtrToStructure<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>(
                    ↳ itemPointer);
356                 itemPointer += size;
357             }
358             return buffer;
359         }
360     }
361     finally
362     {
363         Marshal.FreeHGlobal(pointer);
364     }
365     return null;
366 }

```

```
367     }
368 }
```

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

```
1  using Xunit;
2
3  namespace Platform.Hardware.Cpu.Tests
4  {
5      /// <summary>
6      /// <para>
7      /// Represents the tests.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public static class Tests
12     {
13         /// <summary>
14         /// <para>
15         /// Tests that test.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         [Fact]
20         public static void Test() => Assert.NotEqual(0, CacheLine.Size);
21     }
22 }
```


Index

- ./csharp/Platform.Hardware.Cpu.Tests/CacheLineTests.cs, 8
- ./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, 3