```
LinksPlatform's Platform.Hardware.Cpu Class Library
     ./csharp/Platform.Hardware.Cpu/CacheLine.cs
   using System;
   using System.Runtime.InteropServices;
   namespace Platform. Hardware. Cpu
4
5
        /// <summary>
6
        /// <para>Contains constants related to CPUs cache line.</para>
        /// <para>Содержит константы, относящиеся к строке кэша ЦП.</para>
       public static class CacheLine
{
        /// </summary>
10
11
            /// <summary>
12
            /// <para>Gets the size of CPUs cache line in bytes.</para>
13
            /// <para>Получает размер строки кэша ЦП в байтах.</para>
14
            /// </summary>
            public static readonly int Size = GetSize();
16
17
            /// <summary>
18
            /// <para>
19
            /// Gets the size.
20
            /// </para>
            /// <para></para>
22
            /// </summary>
23
            /// <exception cref="NotSupportedException">
            /// <para>Unrecognized OS platform.</para>
25
            /// <para></para>
26
            /// </exception>
            /// <returns>
            /// <para>The int</para>
29
            /// <para></para>
30
            /// </returns>
31
            private static int GetSize()
32
33
                if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
                    return Windows.GetSize();
36
37
                   (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
                {
39
                    return Linux.GetSize();
40
41
                if (RuntimeInformation.IsOSPlatform(OSPlatform.OSX))
42
43
                    return OSX.GetSize();
44
                throw new NotSupportedException("Unrecognized OS platform.");
46
            }
47
       }
48
   }
49
    ./csharp/Platform.Hardware.Cpu/Linux.cs
   using System;
   using System.Runtime.InteropServices;
2
   #pragma warning disable IDE1006 // Naming Styles
4
   namespace Platform.Hardware.Cpu
6
7
       /// <summary>
       /// <para>
9
        /// Represents the linux.
10
        /// </para>
        /// <para></para>
12
        /// </summary>
13
        internal static class Linux
14
15
            /// <summary>
16
            /// <para>
            /// Gets the size.
18
            /// </para>
19
            /// <para></para>
20
            /// </summary>
21
            /// <returns>
22
            /// <para>The int</para>
            /// <para></para>
            /// </returns>
```

```
public static int GetSize() => (int)sysconf(_SC_LEVEL1_DCACHE_LINESIZE);
26
27
            /// <summary>
28
            /// <para>
            /// Sysconfs the name.
            /// </para>
31
            /// <para></para>
32
            /// </summary>
33
            /// <param name="name">
34
            /// <para>The name.</para>
35
            /// <para></para>
            /// </param>
            /// <returns>
/// <para>The int 64</para>
38
39
            /// <para></para>
/// </returns>
40
41
            [DllImport("libc")]
42
            private static extern Int64 sysconf(Int32 name);
44
            /// <summary>
45
            /// <para>
46
            /// The sc level1 dcache linesize.
47
            /// </para>
48
            /// <para></para>
            /// </summary>
50
51
            private const Int32 _SC_LEVEL1_DCACHE_LINESIZE = 190;
        }
52
   }
53
     ./csharp/Platform.Hardware.Cpu/OSX.cs
1.3
   using System;
   using System.Runtime.InteropServices;
3
   #pragma warning disable IDE1006 // Naming Styles
4
   namespace Platform.Hardware.Cpu
        /// <summary>
8
        /// <para>
9
        /// Represents the osx.
10
        /// </para>
11
        /// <para></para>
12
        /// </summary>
        internal static class OSX
14
15
            /// <summary>
16
            /// <para>
17
            /// Gets the size.
18
            /// </para>
            /// <para></para>
20
            /// </summary>
/// <returns>
21
22
            /// <para>The int</para>
23
            /// <para></para>
24
            /// </returns>
25
            public static int GetSize()
27
                 var sizeOfLineSize = (IntPtr)IntPtr.Size;
28
                 sysctlbyname("hw.cachelinesize", out IntPtr lineSize, ref sizeOfLineSize,
                 → IntPtr.Zero, IntPtr.Zero);
                 return lineSize.ToInt32();
30
            }
31
            /// <summary>
33
            /// <para>
34
            /// Sysctlbynames the name.
35
            /// </para>
            /// <para></para>
37
            /// </summary>
38
            /// <param name="name">
            /// <para>The name.</para>
40
            /// <para></para>
41
            /// </param>
42
            /// <param name="oldp">
43
            /// <para>The oldp.</para>
44
            /// <para></para>
45
            /// </param>
            /// <param name="oldlenp">
```

```
/// <para>The oldlenp.</para>
48
            /// <para></para>
            /// </param>
50
            /// <param name="newp">
            /// <para>The newp.</para>
            /// <para></para>
            /// </param>
54
            /// <param name="newlen">
55
            /// <para>The newlen.</para>
            /// <para></para>
57
            /// </param>
58
            /// <returns>
            /// <para>The int 32</para>
            /// <para></para>
/// </returns>
61
62
            [DllImport("libc")]
            private static extern Int32 sysctlbyname(string name, out IntPtr oldp, ref IntPtr
64
             → oldlenp, IntPtr newp, IntPtr newlen);
        }
65
   }
66
    ./csharp/Platform.Hardware.Cpu/Windows.cs
1.4
   using System;
   using System.Linq;
   using System.Runtime.InteropServices;
3
   #pragma warning disable 0649
   #pragma warning disable IDE0044 // Add readonly modifier
   namespace Platform. Hardware. Cpu
10
        /// <summary>
        /// <para>
11
        /// Represents the windows.
12
        /// </para>
13
        /// <para></para>
14
        /// </summary>
15
        internal static class Windows
17
            /// <summary>
18
            /// <para>
19
            /// Gets the size.
20
            /// </para>
21
            /// <para></para>
            /// </summary>
            /// <exception cref="InvalidOperationException">
24
            /// <para>Could not retrieve the cache line size.</para>
25
            /// <para></para>
            /// </exception>
27
            /// <returns>
28
            /// <para>The int</para>
            /// <para></para>
            /// </returns>
31
32
            public static int GetSize()
                var info = ManagedGetLogicalProcessorInformation();
34
                if (info == null)
35
                {
                     throw new InvalidOperationException("Could not retrieve the cache line size.");
37
38
                return info.First(x => x.Relationship == LOGICAL_PROCESSOR_RELATIONSHIP.RelationCach_
39
                    e).ProcessorInformation.Cache.LineSize;
41
            // http://stackoverflow.com/a/6972620/232574
            /// <summary>
43
            /// <para>
44
            /// The processorcore.
45
            /// </para>
            /// <para></para>
47
            /// </summary>
48
            [StructLayout(LayoutKind.Sequential)]
            struct PROCESSORCORE
50
51
                /// <summary>
                /// <para>
53
                /// The flags.
54
                /// </para>
```

```
/// <para></para>
/// </summary>
    public byte Flags;
/// <summary>
/// <para>
/// The numanode.
/// </para>
/// <para></para>
/// </summary>
[StructLayout(LayoutKind.Sequential)]
struct NUMANODE
    /// <summary>
    /// <para>
    /// The node number.
    /// </para>
    /// <para></para>
    /// </summary>
    public uint NodeNumber;
}
/// <summary>
/// <para>
/// \overline{\text{The}} processor cache type enum.
/// </para>
/// <para></para>
/// </summary>
enum PROCESSOR_CACHE_TYPE
    /// <summary>
    /// <para>
    /// The cache unified processor cache type.
    /// </para>
/// <para>
/// <para>
/// </summary>
    CacheUnified,
    /// <summary>
    /// <para>
    /// The cache instruction processor cache type.
    /// </para>
    /// <para></para>
    /// </summary>
    CacheInstruction,
    /// <summary>
/// <para>
/// The cache data processor cache type.
    /// </para>
    /// <para></para>
/// </summary>
    CacheData,
    /// <summary>
/// <para>
    /// The cache trace processor cache type.
    /// </para>
    /// <para></para>
    /// </summary>
    CacheTrace
}
/// <summary>
/// <para> /// The cache descriptor.
/// </para>
/// <para></para>
/// </summary>
[StructLayout(LayoutKind.Sequential)]
struct CACHE_DESCRIPTOR
    /// <summary>
    /// <para>
    /// The level.
    /// </para>
    /// <para></para>
    /// </summary>
    public Byte Level;
    /// <summary>
    /// <para>
```

56 57

58

60

61

62

63

64

65

66

67

69

70

71

72

73

75

76

78

79

80

81

84

85 86

87

89

90 91 92

93

95

96

97

99

101 102 103

104

105 106

107

108

110

111

114

115 116

119

120

121

122

123

124 125

126

127

128

129

130

131

132

133

134

```
/// The associativity.
    /// </para>
    /// <para></para>
    /// </summary>
    public Byte Associativity;
    /// <summary>
/// <para>
    /// The line size.
    /// </para>
    /// <para></para>
    /// </summary>
    public UInt16 LineSize;
    /// <summary>
/// <para>
    /// The size.
    /// </para>
    /// <para></para>
    /// </summary>
public UInt32 Size;
    /// <summary>
    /// <para>
    /// The type.
    /// </para>
    /// <para></para>
    /// </summary
    public PROCESSOR_CACHE_TYPE Type;
}
/// <summary>
/// <para>
/// The system logical processor information union.
/// </para>
/// <para></para>
/// </summary>
[StructLayout(LayoutKind.Explicit)]
struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION
    /// <summary>
    /// <para>
    /// The processor core.
    /// </para>
    /// <para></para>
/// </summary>
    [FieldOffset(0)]
    public PROCESSORCORE ProcessorCore;
    /// <summary>
    /// <para>
/// The numa node.
/// </para>
/// <para>
/// <para></para>
    /// </summary>
    [FieldOffset(0)]
    public NUMANODE NumaNode;
    /// <summary>
/// <para>
    /// The cache.
    /// </para>
    /// <para></para>
    /// </summary>
    [FieldOffset(0)]
    public CACHE_DESCRIPTOR Cache;
    /// <summary>
    /// <para>
    /// The reserved.
    /// </para>
    /// <para></para>
    /// </summary>
    [FieldOffset(0)]
    private UInt64 Reserved1;
    /// <summary>
    /// <para>
    /// The reserved.
    /// </para>
/// <para></para>
    /// </summary>
    [FieldOffset(8)]
    private UInt64 Reserved2;
}
```

135

137

138

139

140 141

142

143

144

146

147 148

149

150

152 153

154

155

156

158

159

160

161

163

164

165

166

167

170 171

172

173

174

175

176 177

179

185

186

187

188 189

190

191

192

194

196

197

198

200

201

202

203

204

205

206

207

209

210

212

```
/// <summary>
214
              /// <para>
215
              /// \overline{\text{The}} logical processor relationship enum.
216
              /// </para>
217
              /// <para></para>
              /// </summary>
219
              enum LOGICAL_PROCESSOR_RELATIONSHIP
220
221
                   /// <summary>
222
                   /// <para>
223
                   /// \bar{\text{The}} relation processor core logical processor relationship.
                   /// </para>
225
                   /// <para></para>
/// </summary>
226
227
                   RelationProcessorCore,
228
                   /// <summary>
229
                   /// <para>
                   /// The relation numa node logical processor relationship.
231
                   /// </para>
232
                   /// <para></para>
/// </summary>
233
234
                   RelationNumaNode,
235
                   /// <summary>
                   /// <para> /// The relation cache logical processor relationship.
237
238
                   /// </para>
239
                   /// <para></para>
^{240}
                   /// </summary>
241
                   RelationCache,
242
                   /// <summary>
/// <para>
243
244
                   ^{\prime\prime}/^{\prime}/ The relation processor package logical processor relationship.
245
                   /// </para>
^{246}
                   /// <para></para>
247
                   /// </summary>
248
                   RelationProcessorPackage,
249
                   /// <summary>
/// <para>
250
                   ^{\prime\prime\prime}/ The relation group logical processor relationship.
252
                   /// </para>
253
                   /// <para></para>
254
                   /// </summary>
                   RelationGroup,
256
                   /// <summary>
257
                   /// <para>
258
                   /// The relation all logical processor relationship.
259
                   /// </para>
                   /// <para></para>
261
                   /// </summary>
262
                   RelationAll = Oxffff
263
              }
264
              /// <summary>
266
              /// <para>
267
              ^{\prime\prime\prime} The system logical processor information.
268
              /// </para>
269
              /// <para></para>
270
              /// </summary>
271
              private struct SYSTEM_LOGICAL_PROCESSOR_INFORMATION
272
273
                   /// <summary>
                   /// <para>
275
                   /// The processor mask.
276
                   /// </para>
277
                   /// <para></para>
278
                   /// </summary>
279
                   public UIntPtr ProcessorMask;
                   /// <summary>
281
                   /// <para>
282
                   /// The relationship.
283
                   /// </para>
284
                   /// <para></para>
285
                   /// </summary
286
                   public LOGICĂL_PROCESSOR_RELATIONSHIP Relationship;
287
                   /// <summary>
288
289
                   /// <para>
                   /// The processor information.
290
                   /// </para>
291
```

```
/// <para></para>
292
                 /// </summary
                 public SYSTEM_LOGICAL_PROCESSOR_INFORMATION_UNION ProcessorInformation;
294
             }
296
             /// <summary>
297
             /// <para>
             /// Determines whether get logical processor information.
299
             /// </para>
300
             /// <para></para>
             /// </summary>
302
             /// <param name="Buffer">
303
             /// <para>The buffer.</para>
304
             /// <para></para>
             /// </param>
306
             /// <param name="ReturnLength">
307
             /// <para>The return length.</para>
             /// <para></para>
309
             /// </param>
310
             /// <returns>
311
             /// <para>The bool</para>
312
             /// <para></para>
313
             /// </returns>
314
             [DllImport(@"kernel32.dll", SetLastError = true)]
             private static extern bool GetLogicalProcessorInformation(IntPtr Buffer, ref UInt32
316

→ ReturnLength);

317
             /// <summary>
             /// <para>
319
             /// The error insufficient buffer.
320
             /// </para>
             /// <para></para>
322
             /// </summary>
323
             private const int ERROR_INSUFFICIENT_BUFFER = 122;
324
325
             /// <summary>
             /// <para>
327
             /// Manageds the get logical processor information.
328
             /// </para>
             /// <para></para>
330
             /// </summary>
331
             /// <returns>
332
             /// <para>The system logical processor information array</para>
             /// <para></para>
334
             /// </returns>
335
             private static SYSTEM_LOGICAL_PROCESSOR_INFORMATION[]
                 ManagedGetLogicalProcessorInformation()
337
                 var ReturnLength = Ou;
338
                 GetLogicalProcessorInformation(IntPtr.Zero, ref ReturnLength);
                 if (Marshal.GetLastWin32Error() != ERROR_INSUFFICIENT_BUFFER)
340
341
342
                      return null;
                 }
343
                 var pointer = Marshal.AllocHGlobal((int)ReturnLength);
344
345
                 try
346
                      if (GetLogicalProcessorInformation(pointer, ref ReturnLength))
348
                          var size = Marshal.SizeOf<SYSTEM_LOGICAL_PROCESSOR_INFORMATION>();
349
                          var length = (int)ReturnLength / size;
350
                          var buffer = new SYSTEM_LOGICAL_PROCESSOR_INFORMATION[length];
351
352
                          var itemPointer = pointer;
                          for (int i = 0; i < length; i++)</pre>
353
354
                              buffer[i] = Marshal.PtrToStructure<SYSTEM_LOGICAL_PROCESSOR_INFORMATION> |
355
                                   (itemPointer);
                              itemPointer += size;
356
357
                          return buffer;
358
                      }
359
360
                 finally
361
362
                      Marshal.FreeHGlobal(pointer);
363
                 }
364
                 return null;
365
             }
```

```
367
    }
368
1.5 ./csharp/Platform.Hardware.Cpu.Tests/CacheLineTests.cs
    using Xunit;
    namespace Platform.Hardware.Cpu.Tests
         /// <summary>
/// <para>
/// Represents the tests.
 5
 6
         /// </para>
         /// <para></para>
         .., \ \summary>
public static class Tests
{
11
12
              /// <summary>
/// <para>
13
14
              /// Tests that test.
15
              /// </para>
              /// <para></para>
/// </summary>
17
18
               [Fact]
19
              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