

LinksPlatform's Platform.Singletons Class Library

1.1 ./csharp/Platform.Singletons/Default[T].cs

```
1 using System;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable RECS0017 // Possible compare of value type with 'null'
5
6 namespace Platform.Singletons
7 {
8     /// <summary>
9     /// <para>Represents an access point to instances of default types (created using the
10     ↪ constructor with no arguments).</para>
11     /// <para>Представляет собой точку доступа к экземплярам типов по умолчанию (созданных с
12     ↪ помощью конструктора без аргументов).</para>
13     /// </summary>
14     /// <typeparam name="T"><para>The type of instance of the object.</para><para>Тип экземпляра
15     ↪ объекта.</para></typeparam>
16     public static class Default<T>
17     where T : new()
18     {
19         /// <summary>
20         /// <para>
21         /// The thread instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         [ThreadStatic]
26         private static T _threadInstance;
27
28         /// <summary>
29         /// <para>Returns an instance of an object by default.</para>
30         /// <para>Возвращает экземпляр объекта по умолчанию.</para>
31         /// </summary>
32         public static readonly T Instance = new T();
33
34         /// <summary>
35         /// <para>If you really need maximum performance, use this property. This property
36         ↪ should create only one instance per thread.</para>
37         /// <para>Если вам действительно нужна максимальная производительность, используйте это
38         ↪ свойство. Это свойство должно создавать только один экземпляр на поток.</para>
39         /// </summary>
40         /// <remarks>
41         /// <para>Check for null is intended to create only classes, not structs.</para>
42         /// <para>Проверка на значение null выполняется специально для создания только классов,
43         ↪ а не структур.</para>
44         /// </remarks>
45         public static T ThreadInstance
46         {
47             [MethodImpl(MethodImplOptions.AggressiveInlining)]
48             get => _threadInstance == null ? _threadInstance = new T() : _threadInstance;
49             ↪ //-V3111
50         }
51     }
52 }
```

1.2 ./csharp/Platform.Singletons/Global.cs

```
1 using System.Runtime.CompilerServices;
2
3 namespace Platform.Singletons
4 {
5     /// <summary>
6     /// <para>Contains the global state of the system.</para>
7     /// <para>Содержит глобальное состояние системы.</para>
8     /// </summary>
9     public static class Global
10     {
11         /// <summary>
12         /// <para>
13         /// Represents a garbage field where you can dump unnecessary values.
14         /// In some cases, this may help to avoid unwanted optimization and pretend that the
15         ↪ value is really used.
16         /// This may be useful when implementing performance tests.
17         /// </para>
18         /// <para>
19         /// Представляет поле-помойку, куда можно сбрасывать ненужные значения.
20         /// В некоторых случаях это может помочь избежать нежелательной оптимизации и сделать
21         ↪ вид, что значение действительно используется.
22         /// Такое может быть полезно при реализации тестов на производительность.
23         
```

```

21     /// </para>
22     /// </summary>
23     public static object Trash
24     {
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         get;
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         set;
29     }
30 }
31 }

```

1.3 ./csharp/Platform.Singletons/Singleton.cs

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Singletons
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the singleton.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public static class Singleton
16     {
17         /// <summary>
18         /// <para>
19         /// Creates the creator.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <typeparam name="T">
24         /// <para>The .</para>
25         /// <para></para>
26         /// </typeparam>
27         /// <param name="creator">
28         /// <para>The creator.</para>
29         /// <para></para>
30         /// </param>
31         /// <returns>
32         /// <para>A singleton of t</para>
33         /// <para></para>
34         /// </returns>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public static Singleton<T> Create<T>(Func<T> creator) => new Singleton<T>(creator);
37
38         /// <summary>
39         /// <para>
40         /// Creates the factory.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         /// <typeparam name="T">
45         /// <para>The .</para>
46         /// <para></para>
47         /// </typeparam>
48         /// <param name="factory">
49         /// <para>The factory.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>A singleton of t</para>
54         /// <para></para>
55         /// </returns>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         public static Singleton<T> Create<T>(IFactory<T> factory) => new
58             ↳ Singleton<T>(factory.Create);
59
60         /// <summary>
61         /// <para>
62         /// Gets the creator.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <typeparam name="T">

```

```

66     /// <para>The .</para>
67     /// <para></para>
68     /// </typeparam>
69     /// <param name="creator">
70     /// <para>The creator.</para>
71     /// <para></para>
72     /// </param>
73     /// <returns>
74     /// <para>The</para>
75     /// <para></para>
76     /// </returns>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     public static T Get<T>(Func<T> creator) => Create(creator).Instance;
79
80     /// <summary>
81     /// <para>
82     /// Gets the factory.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <typeparam name="T">
87     /// <para>The .</para>
88     /// <para></para>
89     /// </typeparam>
90     /// <param name="factory">
91     /// <para>The factory.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>The</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public static T Get<T>(IFactory<T> factory) => Create(factory).Instance;
100 }
101 }

```

1.4 ./csharp/Platform.Singletons/Singleton[T].cs

```

1  using System;
2  using System.Collections.Concurrent;
3  using System.Reflection;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Lists;
6  using Platform.Reflection;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9  #pragma warning disable RECS0108 // Warns about static fields in generic types
10
11 namespace Platform.Singletons
12 {
13     /// <summary>
14     /// <para>
15     /// The singleton.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     public struct Singleton<T>
20     {
21         private static readonly ConcurrentDictionary<Func<T>, byte[]> _functions = new
22             ↳ ConcurrentDictionary<Func<T>, byte[]>();
23         private static readonly ConcurrentDictionary<byte[], T> _singletons = new
24             ↳ ConcurrentDictionary<byte[], T>(Default<IListEqualityComparer<byte[]>>.Instance);
25
26         /// <summary>
27         /// <para>
28         /// Gets the instance value.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         public T Instance
33         {
34             [MethodImpl(MethodImplOptions.AggressiveInlining)]
35             get;
36         }
37
38         /// <summary>
39         /// <para>
40         /// Initializes a new <see cref="Singleton"/> instance.
41         /// </para>

```

```

40     /// <para></para>
41     /// </summary>
42     /// <param name="creator">
43     /// <para>A creator.</para>
44     /// <para></para>
45     /// </param>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public Singleton(Func<T> creator) => Instance =
        ↪ _singletons.GetOrAdd(_functions.GetOrAdd(creator,
        ↪ creator.GetMethodInfo().GetILBytes()), key => creator());
48 }
49 }

```

1.5 ./csharp/Platform.Singletons.Tests/DefaultTests.cs

```

1 using Xunit;
2
3 namespace Platform.Singletons.Tests
4 {
5     /// <summary>
6     /// <para>
7     /// Represents the default tests.
8     /// </para>
9     /// <para></para>
10    /// </summary>
11    public class DefaultTests
12    {
13        /// <summary>
14        /// <para>
15        /// Tests that struct instance test.
16        /// </para>
17        /// <para></para>
18        /// </summary>
19        [Fact]
20        public void StructInstanceTest()
21        {
22            Assert.Equal(0, Default<int>.Instance);
23        }
24
25        /// <summary>
26        /// <para>
27        /// Tests that class instance test.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        [Fact]
32        public void ClassInstanceTest()
33        {
34            Assert.NotNull(Default<object>.Instance);
35        }
36
37        /// <summary>
38        /// <para>
39        /// Tests that struct thread instance test.
40        /// </para>
41        /// <para></para>
42        /// </summary>
43        [Fact]
44        public void StructThreadInstanceTest()
45        {
46            Assert.Equal(0, Default<int>.ThreadInstance);
47        }
48
49        /// <summary>
50        /// <para>
51        /// Tests that class thread instance test.
52        /// </para>
53        /// <para></para>
54        /// </summary>
55        [Fact]
56        public void ClassThreadInstanceTest()
57        {
58            Assert.NotNull(Default<object>.ThreadInstance);
59        }
60    }
61 }

```

1.6 ./csharp/Platform.Singletons.Tests/GlobalTests.cs

```

1 using Xunit;
2

```

```

3 namespace Platform.Singletons.Tests
4 {
5     /// <summary>
6     /// <para>
7     /// Represents the global tests.
8     /// </para>
9     /// <para></para>
10    /// </summary>
11    public class GlobalTests
12    {
13        /// <summary>
14        /// <para>
15        /// Tests that trash is null test.
16        /// </para>
17        /// <para></para>
18        /// </summary>
19        [Fact]
20        public void TrashIsNullTest()
21        {
22            Assert.Null(Global.Trash);
23        }
24    }
25 }

```

1.7 ./csharp/Platform.Singletons.Tests/SingletonTests.cs

```

1 using Xunit;
2
3 namespace Platform.Singletons.Tests
4 {
5     /// <summary>
6     /// <para>
7     /// Represents the singleton tests.
8     /// </para>
9     /// <para></para>
10    /// </summary>
11    public class SingletonTests
12    {
13        /// <summary>
14        /// <para>
15        /// Tests that two values are the same test.
16        /// </para>
17        /// <para></para>
18        /// </summary>
19        [Fact]
20        public void TwoValuesAreTheSameTest()
21        {
22            var value1 = Singleton.Get(() => 1);
23            var value2 = Singleton.Get(() => 1);
24            Assert.Equal(value1, value2);
25        }
26
27        // Looks like ILBytes do not help here
28        //[Fact]
29        //public void TwoFunctionsAreTheSameTest()
30        //{
31            //    //Func<Func<int>> factory = () => () => 1;
32            //    var func1 = Singleton.Get<Func<int>>(() => () => 1);
33            //    var func2 = Singleton.Get<Func<int>>(() => () => 1);
34            //    Assert.Equal(func1, func2);
35        //}
36    }
37 }

```

Index

- ./csharp/Platform.Singletons.Tests/DefaultTests.cs, 4
- ./csharp/Platform.Singletons.Tests/GlobalTests.cs, 4
- ./csharp/Platform.Singletons.Tests/SingletonTests.cs, 5
- ./csharp/Platform.Singletons/Default[T].cs, 1
- ./csharp/Platform.Singletons/Global.cs, 1
- ./csharp/Platform.Singletons/Singleton.cs, 2
- ./csharp/Platform.Singletons/Singleton[T].cs, 3