LinksPlatform's Platform.Singletons Class Library

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

```csharp
using System;
using System.Runtime.CompilerServices;

#pragma warning disable RECS0017 // Possible compare of value type with 'null'

namespace Platform.Singletons
{
    /// <summary>
    /// <para>Represents an access point to instances of default types (created using the constructor with no arguments).</para>
    /// <para>Представляет собой точку доступа к экземплярям типов по умолчанию (созданных с помощью конструктора без аргументов).</para>
    /// </summary>
    /// <typeparam name="T"><para>The type of instance of the object.</para><para>Тип экземпляра объекта.</para></typeparam>
    public static class Default<T>
        where T : new()
    {
        /// <summary>
        /// <para>
        /// The thread instance.
        /// </para>
        /// <para></para>
        /// </summary>
        [ThreadStatic]
        private static T _threadInstance;

        /// <summary>
        /// <para>Returns an instance of an object by default.</para>
        /// <para>Возвращает экземпляр объекта по умолчанию.</para>
        /// </summary>
        public static readonly T Instance = new T();

        /// <summary>
        /// <para>If you really need maximum performance, use this property. This property should create only one instance per thread.</para>
        /// <para>Если вам действительно нужна максимальная производительность, используйте это свойство. Это свойство должно создавать только один экземпляр на поток.</para>
        /// </summary>
        /// <remarks>
        /// <para>Check for null is intended to create only classes, not structs.</para>
        /// <para>Проверка на значение null выполняется специально для создания только классов, а не структур.</para>
        /// </remarks>
        public static T ThreadInstance
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _threadInstance == null ? _threadInstance = new T() : _threadInstance; //-V3111
        }
    }
}
```

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

```csharp
using System.Runtime.CompilerServices;

namespace Platform.Singletons
{
    /// <summary>
    /// <para>Contains the global state of the system.</para>
    /// <para>Содержит глобальное состояние системы.</para>
    /// </summary>
    public static class Global
    {
        /// <summary>
        /// <para>
        /// Represents a garbage field where you can dump unnecessary values.
        /// In some cases, this may help to avoid unwanted optimization and pretend that the value is really used.
        /// This may be useful when implementing performance tests.
        /// </para>
        /// <para>
        /// Представляет поле-помойку, куда можно сбрасывать ненужные значения.
        /// В некоторых случаях это может помочь избежать нежелательной оптимизации и сделать вид, что значение действительно используется.
        /// Такое может быть полезно при реализации тестов на производительность.
        /// </para>
        /// </summary>
        public static object Trash
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }
    }
}
```

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

```
using System;
using System.Runtime.CompilerServices;
using Platform.Interfaces;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Singletons
{
    /// <summary>
    /// <para>
    /// Represents the singleton.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class Singleton
    {
        /// <summary>
        /// <para>
        /// Creates the creator.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="T">
        /// <para>The .</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="creator">
        /// <para>The creator.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A singleton of t</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static Singleton<T> Create<T>(Func<T> creator) => new Singleton<T>(creator);

        /// <summary>
        /// <para>
        /// Creates the factory.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="T">
        /// <para>The .</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="factory">
        /// <para>The factory.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A singleton of t</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static Singleton<T> Create<T>(IFactory<T> factory) => new
         ↪  Singleton<T>(factory.Create);

        /// <summary>
        /// <para>
        /// Gets the creator.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="T">
```

```csharp
        /// <para>The .</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="creator">
        /// <para>The creator.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T Get<T>(Func<T> creator) => Create(creator).Instance;

        /// <summary>
        /// <para>
        /// Gets the factory.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="T">
        /// <para>The .</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="factory">
        /// <para>The factory.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static T Get<T>(IFactory<T> factory) => Create(factory).Instance;
    }
}
```

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

```csharp
using System;
using System.Collections.Concurrent;
using System.Reflection;
using System.Runtime.CompilerServices;
using Platform.Collections.Lists;
using Platform.Reflection;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
#pragma warning disable RECS0108 // Warns about static fields in generic types

namespace Platform.Singletons
{
    /// <summary>
    /// <para>
    /// The singleton.
    /// </para>
    /// <para></para>
    /// </summary>
    public struct Singleton<T>
    {
        private static readonly ConcurrentDictionary<Func<T>, byte[]> _functions = new
         ↪  ConcurrentDictionary<Func<T>, byte[]>();
        private static readonly ConcurrentDictionary<byte[], T> _singletons = new
         ↪  ConcurrentDictionary<byte[], T>(Default<IListEqualityComparer<byte>>.Instance);

        /// <summary>
        /// <para>
        /// Gets the instance value.
        /// </para>
        /// <para></para>
        /// </summary>
        public T Instance
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Singleton"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="creator">
        /// <para>A creator.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Singleton(Func<T> creator) => Instance =
         ↪  _singletons.GetOrAdd(_functions.GetOrAdd(creator,
         ↪  creator.GetMethodInfo().GetILBytes()), key => creator());
    }
}
```

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

```
using Xunit;

namespace Platform.Singletons.Tests
{
    /// <summary>
    /// <para>
    /// Represents the default tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class DefaultTests
    {
        /// <summary>
        /// <para>
        /// Tests that struct instance test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void StructInstanceTest()
        {
            Assert.Equal(0, Default<int>.Instance);
        }

        /// <summary>
        /// <para>
        /// Tests that class instance test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void ClassInstanceTest()
        {
            Assert.NotNull(Default<object>.Instance);
        }

        /// <summary>
        /// <para>
        /// Tests that struct thread instance test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void StructThreadInstanceTest()
        {
            Assert.Equal(0, Default<int>.ThreadInstance);
        }

        /// <summary>
        /// <para>
        /// Tests that class thread instance test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void ClassThreadInstanceTest()
        {
            Assert.NotNull(Default<object>.ThreadInstance);
        }
    }
}
```

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

```
using Xunit;

```

```
namespace Platform.Singletons.Tests
{
    /// <summary>
    /// <para>
    /// Represents the global tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class GlobalTests
    {
        /// <summary>
        /// <para>
        /// Tests that trash is null test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void TrashIsNullTest()
        {
            Assert.Null(Global.Trash);
        }
    }
}
```

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

```
using Xunit;

namespace Platform.Singletons.Tests
{
    /// <summary>
    /// <para>
    /// Represents the singleton tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class SingletonTests
    {
        /// <summary>
        /// <para>
        /// Tests that two values are the same test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void TwoValuesAreTheSameTest()
        {
            var value1 = Singleton.Get(() => 1);
            var value2 = Singleton.Get(() => 1);
            Assert.Equal(value1, value2);
        }

        // Looks like ILBytes do not help here
        //[Fact]
        //public void TwoFunctionsAreTheSameTest()
        //{
        //    //Func<Func<int>> factory = () => () => 1;
        //    var func1 = Singleton.Get<Func<int>>(() => () => 1);
        //    var func2 = Singleton.Get<Func<int>>(() => () => 1);
        //    Assert.Equal(func1, func2);
        //}
    }
}
```

Index