1.1 ./csharp/Platform.Random/RandomExtensions.cs

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

namespace Platform.Random
{
    /// <summary>
    /// <para>Contains extension methods for <see cref="System.Random"/> class.</para>
    /// <para>Содержит методы расширения для класса <see cref="System.Random"/>.</para>
    /// </summary>
    public static class RandomExtensions
    {
        /// <summary>
        /// <para>Returns a random 64-bit unsigned integer that is greater than or equal to <see cref="ulong.MinValue"/>, and less than or equal to <see cref="ulong.MaxValue"/>.</para>
        /// <para>Возвращает случайное 64-разрядное целое число без знака, которое больше или равно <see cref="ulong.MinValue"/> и меньше или равно <see cref="ulong.MaxValue"/>.</para>
        /// </summary>
        /// <param name="random"><para>A pseudo-random number generator.</para><para>Генератор псевдослучайных чисел.</para></param>
        /// <returns>
        /// <para>A 64-bit unsigned integer that is greater than or equal to <see cref="ulong.MinValue"/>, and less than or equal to <see cref="ulong.MaxValue"/>.</para>
        /// <para>64-разрядное целое число без знака, которое больше или равно <see cref="ulong.MinValue"/> и меньше или равно <see cref="ulong.MaxValue"/>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong NextUInt64(this System.Random random) => random.NextUInt64(Range.UInt64);

        /// <summary>
        /// <para>Returns a random 64-bit unsigned integer that is greater than or equal to minimum of specified range, and less than or equal to maximum of specified range.</para>
        /// <para>Возвращает случайное 64-разрядное целое число без знака, которое больше или равно минимуму указанного диапазона и меньше или равно максимуму указанного диапазона.</para>
        /// </summary>
        /// <param name="random"><para>A pseudo-random number generator.</para><para>Генератор псевдослучайных чисел.</para></param>
        /// <param name="range"><para>The range of possible values.</para><para>Диапазон возможных значений.</para></param>
        /// <returns>
        /// <para>A 64-bit unsigned integer that is greater than or equal to the minimum of specified range, and less than or equal to the maximum of the specified range.</para>
        /// <para>64-разрядное целое число без знака, которое больше или равно минимуму указанного диапазона и меньше или равно максимуму указанного диапазона.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ulong NextUInt64(this System.Random random, Range<ulong> range) => (ulong)(random.NextDouble() * range.Difference()) + range.Minimum;

        /// <summary>
        /// <para>Return a random <see cref="bool"/> value.</para>
        /// <para>Возвращает случайное значение <see cref="bool"/>.</para>
        /// </summary>
        /// <param name="random"><para>A pseudo-random number generator.</para><para>Генератор псевдослучайных чисел.</para></param>
        /// <returns><para>A random <see cref="bool"/> value.</para><para>Случайное значение <see cref="bool"/>.</para></returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool NextBoolean(this System.Random random) => random.Next(2) == 1;
    }
}
```

1.2 ./csharp/Platform.Random/RandomHelpers.cs

```
namespace Platform.Random
{
    /// <summary>
    /// <para>Contains field-helper for <see cref="System.Random"/> class.</para>
    /// <para>Содержит вспомогательное поле для класса <see cref="System.Random"/>.</para>
    /// </summary>
    public static class RandomHelpers
```

```
    {
        /// <summary>
        /// <para>Returns the pseudorandom number generator that is using the time of the first access to this field as seed.</para>
        /// <para>Возвращает генератор псевдослучайных чисел использующий в качестве seed время первого обращения к этому полю.</para>
        /// </summary>
        public static readonly System.Random Default = new System.Random(System.DateTime.UtcNow.Ticks.GetHashCode());
    }
}
```

1.3 ./csharp/Platform.Random.Tests/RandomExtensionsTests.cs

```
using Xunit;

namespace Platform.Random.Tests
{
    /// <summary>
    /// <para>
    /// Represents the random extensions tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class RandomExtensionsTests
    {
        /// <summary>
        /// <para>
        /// Tests that next u int 64 test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void NextUInt64Test()
        {
            var lastValue = 0UL;
            var theSameCount = 0;
            for (var i = 0; i < 10; i++)
            {
                var newValue = RandomHelpers.Default.NextUInt64();
                if (newValue == lastValue)
                {
                    theSameCount++;
                }
                else
                {
                    lastValue = newValue;
                    theSameCount = 0;
                }
                Assert.InRange(RandomHelpers.Default.NextUInt64((0UL, 5UL)), 0UL, 5UL);
            }
            Assert.True(theSameCount < 8);
        }

        /// <summary>
        /// <para>
        /// Tests that next boolean test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void NextBooleanTest()
        {
            var trueCount = 0;
            var falseCount = 0;
            for (var i = 0; i < 10; i++)
            {
                var newValue = RandomHelpers.Default.NextBoolean();
                if (newValue)
                {
                    trueCount++;
                }
                else
                {
                    falseCount++;
                }
            }
            Assert.True(trueCount > 0);
            Assert.True(falseCount > 0);
        }
```

```
    }
}
```

1.4 ./csharp/Platform.Random.Tests/RandomHelpersTests.cs

```
using Xunit;

namespace Platform.Random.Tests
{
    /// <summary>
    /// <para>
    /// Represents the random helpers tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class RandomHelpersTests
    {
        /// <summary>
        /// <para>
        /// Tests that default field test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void DefaultFieldTest()
        {
            Assert.NotNull(RandomHelpers.Default);
        }
    }
}
```

Index