

Unsafe Context

```
/*
A Programmer's Introduction to C# (Second Edition)
by Eric Gunnerson

Publisher: Apress L.P.
ISBN: 1-893115-62-3
*/
// 36 – Deeper into C#Unsafe Context
// copyright 2000 Eric Gunnerson
// file=unsafe.cs
// compile with: csc /unsafe /o+ unsafe.cs
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;

public class UnsafeContext
{
    const int iterations = 20000; // # to do copy
    const int points = 1000; // # of points in array
    const int retryCount = 5; // # of times to retry

    public delegate Point[] CloneFunction(Point[] a);

    public static void TimeFunction(Point[] arr,
        CloneFunction func, string label)
    {
        Point[] arrCopy = null;
        long start;
        long delta;
        double min = 5000.0d; // big number;

        // do the whole copy retryCount times, find fastest time
        for (int retry = 0; retry < retryCount; retry++) { start =
            Counter.Value; for (int iterate = 0; iterate < iterations;
            iterate++) arrCopy = func(arr); delta = Counter.Value - start;
```

```

double result = (double) delta / Counter.Frequency; if (result
< min) min = result; } Console.WriteLine("{0}: {1:F3}
seconds", label, min); } public static void Main() {
Console.WriteLine("Points, Iterations: {0} {1}", points,
iterations); Point[] arr = new Point[points]; for (int index =
0; index < points; index++) arr[index] = new Point(3, 5);
TimeFunction(arr, new
CloneFunction(Point.ClonePointArrayMemcpy), "Memcpy");
TimeFunction(arr, new
CloneFunction(Point.ClonePointArrayUnsafe), "Unsafe");
TimeFunction(arr, new CloneFunction(Point.ClonePointArray),
"Baseline"); } } class Counter { public static long Frequency
{ get { long freq = 0; QueryPerformanceFrequency(ref freq);
return freq; } } public static long Value { get { long count =
0; QueryPerformanceCounter(ref count); return count; } }
[System.Runtime.InteropServices.DllImport("KERNEL32",
CharSet=System.Runtime.InteropServices.CharSet.Auto)] private
static extern bool QueryPerformanceCounter( ref long
lpPerformanceCount);
[System.Runtime.InteropServices.DllImport("KERNEL32",
CharSet=System.Runtime.InteropServices.CharSet.Auto)] private
static extern bool QueryPerformanceFrequency( ref long
lpFrequency); } public struct Point { public Point(int x, int
y) { this.x = x; this.y = y; } // safe version public static
Point[] ClonePointArray(Point[] a) { Point[] ret = new
Point[a.Length]; for (int index = 0; index < a.Length;
index++) ret[index] = a[index]; return(ret); } // unsafe
version using pointer arithmetic unsafe public static Point[]
ClonePointArrayUnsafe(Point[] a) { Point[] ret = new
Point[a.Length]; // a and ret are pinned; they cannot be moved
by // the garbage collector inside the fixed block. fixed
(Point* src = a, dest = ret) { Point* pSrc = src; Point* pDest
= dest; for (int index = 0; index < a.Length; index++) {
*pDest = *pSrc; pSrc++; pDest++; } } return(ret); } // import
CopyMemory from kernel32 [DllImport("kernel32.dll")] unsafe
public static extern void CopyMemory(void* dest, void* src,
int length); // unsafe version calling CopyMemory() unsafe

```

```
public static Point[] ClonePointArrayMemcpy(Point[] a) {  
    Point[] ret = new Point[a.Length]; fixed (Point* src = a, dest  
    = ret) { CopyMemory(dest, src, a.Length * sizeof(Point)); }  
    return(ret); } public override string ToString() {  
    return(String.Format("({0}, {1})", x, y)); } int x; int y; }  
[/csharp]
```