
Object Pinning and GC Handles

So far I have implied that the SOH is better managed than the LOH because it compacts and doesn't suffer from fragmentation issues (as mentioned in Chapter 2's *LOH* section). Well, that's not entirely true.

If you want to make calls from your .NET application to other unmanaged application APIs or COM objects, you will probably want to pass data to them. If the data you pass is allocated on the heap, then it could be moved at some point during compaction as part of GC.

This is not a problem for .NET applications, but it's a huge problem for unmanaged apps which rely on fixed locations for objects. So we need a way of fixing the memory location of the objects we pass to unmanaged calls.

GC Handles

.NET uses a structure called `GCHandle` to keep track of heap objects, which can also be used to pass object references between managed and unmanaged domains. .NET maintains a table of `GCHandles` to achieve this. A `GCHandle` can be one of four types:

- **Normal**–tracks standard heap objects
- **Weak**–used to track short weak references
- **Weak Track Resurrection**–used for long weak references
- **Pinned**–used to fix an object at a specific address in memory.

Object pinning

We can create GC Handles directly in code, and so can use them to create pinned references to objects we need to pass to unmanaged code. Listing 3.11 illustrates the use of creating a pinned handle to an object, which can then be passed to an unmanaged call.

```
byte[] buffer = new byte[512];

GCHandle h = GCHandle.Alloc(buffer, GCHandleType.Pinned);

IntPtr ptr = h.AddrOfPinnedObject();

// Call native API and pass buffer

if (h.IsAllocated) h.Free();
```

Listing 3.11: Using `GCHandle` to pin an object in memory.

Notice how the handle is allocated using `Alloc`, and then `Free` is called once the call to the API is complete. Objects will also be pinned if you use a `fixed` block.

```
unsafe static void Main()
{
        Person p = new Person();
```

```
        p.age = 25;
        // Pin p
        fixed (int* a = &p.age)
        {
            // Do something
        }
        // p unpinned
}
```

Listing 3.12: Pinning using `fixed` in an unsafe block.

Problems with object pinning

The main problem with object pinning is that it can cause SOH fragmentation. If an object is pinned during a GC then, by definition, it can't be relocated. Depending on how you use pinning, it can reduce the efficiency of compaction, leaving gaps in the heap.

The best advice is to pin for a very short time and then release, to minimize compaction issues.