# How to allocate address space with a custom alignment or in a custom address region

devblogs.microsoft.com/oldnewthing/20231229-00

December 29, 2023

Raymond Chen

Windows 10 version 1803 added the ability to request specific alignment and address region requirements when creating address space mappings either for virtual memory (`Virtual-Alloc2`) or file mappings (`MapViewOfFile3`). These new functions let you pass a `MEM_ADDRESS_REQUIREMENTS` structure which can specify additional constraints:

- `LowestStartingAddress`: The lowest address in the region.
- `HighestEndingAddress`: The highest address in the region.
- `Alignment`: The desired alignment.

If any field is zero, then the system uses whatever value it normally would have. For example, if you say that the alignment is zero, then the memory will be aligned on the allocation granularity (for normal-sized pages) or the large page granularity (for large pages).

Here are some sample usages:

| Scenario | LowestStarting-Address | HighestEnding-Address | Alignment |
|---|---|---|---|
| Align to multiple of *N* | 0 | 0 | *N* |
| Below 4GB | 0 | `0x00000000`FFFFFFFF` | 0 |
| Above 4GB | `0x00000001`00000000` | 0 | 0 |
| At 1MB boundary between 2GB and 4GB | `0x00000000`80000000` | `0x00000000`FFFFFFFF` | `0x00000000`00100000` |

Note that the `HighestEndingAddress` is endpoint-inclusive. In other words, it is the highest address in the region, not the highest address *plus one*.

Here's how you pass the `MEM_ADDRESS_REQUIREMENTS`:

```
MEM_ADDRESS_REQUIREMENTS requirements = {0};

requirements.LowestStartingAddress = ⟦ value ⟧;
requirements.HighestStartingAddress = ⟦ value ⟧;
requirements.Alignment = ⟦ value ⟧;

MEM_EXTENDED_PARAMETER param = {0};
param.Type = MemExtendedParameterAddressRequirements;
param.Pointer = &requirements;

auto result = VirtualAlloc2(nullptr, nullptr,
        size, MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE,
        &param, 1);
```

The `VirtualAlloc2` function takes a pointer to an array of `MEM_EXTENDED_PARAMETER` structures, but since we have only one, we just pass the address of a single object and tell the system that we have an array of only one element.

Note that you cannot combine non-default `MEM_ADDRESS_REQUIREMENTS` with an explicit `BaseAddress`. If you pass an explicit base address, then you are saying, "I want the memory to be exactly here; I don't want the system to choose for me." Under those conditions, it's meaningless to tell the system "And here is how I'd like you to choose the address for me."