Memory Banks Sample

*This sample is compatible with the November 2015 Xbox One XDK or later*

# Description

This sample demonstrates several different methods available for managing memory banks.

* Random – Baseline memory block allocated through VirtualAlloc.
* Fixed – Demonstrates how to allocate memory at a specific memory address.
* Read-Only – Demonstrates how to convert memory to read-only.
* Bank Switching – Demonstrates how to create a chain of memory banks that can be rotated.
* Shared – Demonstrates how to create multiple memory banks that share the same physical location.

# Using the sample

Press the corresponding button on the controller for each demonstration. It will report success or failure.

# Implementation notes

## Random

This demonstration serves as the baseline for the other demonstrations. It allocates a block of memory using VirtualAlloc and then reads a binary tree from disk into the memory block. The data from the disk needs to be fixed up. Internally the binary tree stores indexes into the memory block it’s allocated within. All of the internal pointers for the binary tree need to be updated based on the index since the base address changes between runs of the program.

## Fixed

This demonstration works the same way as the random address demonstration. The main difference is that the base address is stored in the data file. This allows the program to recreate the same block at the same memory location for each run of the program. This means the internal indexes are not needed and there is no need for pointer fixup. The data can be loaded from disk much faster with almost zero CPU overhead.

## Bank Switching

This demonstration shows how to create multiple memory banks that can be switched between. This swaps the virtual address for two blocks of physical memory. More than two memory banks could be used and then rotated between. Several uses for this is in the creation of logging data, frame data used to save replays, and other places memory is moved between buffers. This allows the removal of the memory move operation and results in dramatically improved performance.

## Shared

This demonstration shows how to create multiple memory blocks that all point to the same physical block. If two virtual blocks are created adjacent to each other this allows a ring buffer that doesn’t require multiple memory copy operations for boundary copies. As data is written across the end of one virtual block it will automatically wrap to the start of the physical block.

Another use is to create multiple virtual pointers with different permissions, such as read-write and read-only. Care must be taken though in mixing certain types of permissions that adjust cache usage, for example write-combine and cacheable. This will result in the CPU caching the data in different ways when accessing one pointer or the other that could create hard to track down bugs.

## Read-Only

This demonstration shows how to create read-only pages. It does this using the shared demonstration where one block is marked read-write and another block is marked read-only. A prime use for this pattern is to track down random memory corruption. For example, static data used by a title. The file loading system could use the read-write address for creating the data. The rest of the title uses the read-only pointer for access. If there is a memory corruption issue it will cause an exception at the exact location in source causing the problem.

# Update history

Initial release October 2016