Data Breakpoints

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

# Description

This sample shows how to create hardware data breakpoints that are useful for detecting different types of memory access. They are handled entirely by the processor core which means they do not effect execution speed. The major downside to using them is there are only 4 slots available on the processor core.

Windows allows only local breakpoints to a thread. This means that a breakpoint will only be active while the thread in question is executing. Since the hardware breakpoints are bound to a thread they will follow the thread as it executes on various cores. If needed each thread can be set with the same breakpoint.

# Contents

## DataBreak.cpp/h

* Standalone package that provides the ability to set and clear hardware data breakpoints.
* SetHardwareBreakPointForThread
  + Sets a breakpoint given the slot in question and address.
  + If the slot is already used it will be overwritten.
* ClearHardwareBreakPointForThread
  + Clears the breakpoint on the specified slot

# Implementation notes

To set hardware breakpoint the contents of the debug registers need to be adjusted. The main problem is that access to these registers is only available to the kernel. The trick to setting the debug registers is to adjust the thread context. This will cause the scheduler to write the contents to the debug register during a context switch.

The context for a thread can only be changed if the thread is suspended. Because of this if the target thread is the current thread another thread is required to perform the operation. The sample will create a temporary worker thread to perform the context switch.

The exception thrown can be caught with either structured exception handling **(\_\_try**, **\_\_except**) or through the unhandled exception filter. However, the capture exception record is ignored if it’s being used with **MiniDumpWriteDump**. However, if **MiniDumpWriteDump** is being called on the thread that fired the exception the offending code will still be in the call stack, just further up. The unhandled exception filter is called in the context of the offending thread.

## Important Note:

The exception thrown by this system is a single step exception which is treated specially by the OS. If a debugger is attached the debugger will catch it first. By default, Visual Studio will ignore it and control will pass to the titles exception handler unless you’re single stepping through the code. By default, KD will break at the line of code causing the exception. If no debugger is attached, however the **EnableKernelDebugging** defines the behavior. If it is enabled the ERA will freeze. The console is waiting for a debugger to be attached. If **EnableKernelDebugging** is not enabled the titles exception handler will be called.

The **DataBreakThread** function in DataBreak.cpp documents the contents needed for the debug registers.

# Update history

Initial release October 2016