

符号代理客户端样本

*此样本与 2015 年 6 月的 Xbox One XDK 或更高版本兼容。*

*兼容性注意：6 月 XDK 中存在一个已知问题，即 xbSymbolProxy 无法正确解析源代码行信息。这个问题在 8 月 XDK 中得到修复，并将包含在 6 月 XDK QFE 1 中*

# 描述

此样本演示了如何使用 SymbolResolve API 解析主机上地址的符号信息。目前 SymbolResolve.h 中有两个 API，GetSymbolFromAddress() 和 GetSourceLineFromAddress()。这些 API 在主机端诊断代码中很有用，例如，用于解码捕获的反向跟踪中的返回地址。如需了解更多详细信息，请参阅 XDK 文档。

样本还在 CallStack.h/.cpp 中提供了一些有用的实用程序代码，用于捕获和解码调用堆栈。用于捕获反向跟踪的三个辅助函数：

捕获当前线程的反向跟踪

CaptureBackTraceFromCurrentThread()

从提供的上下文记录开始捕获反向跟踪。

CaptureBackTraceFromContext()

从提供的上下文记录开始计算调用堆栈中的堆栈帧数。

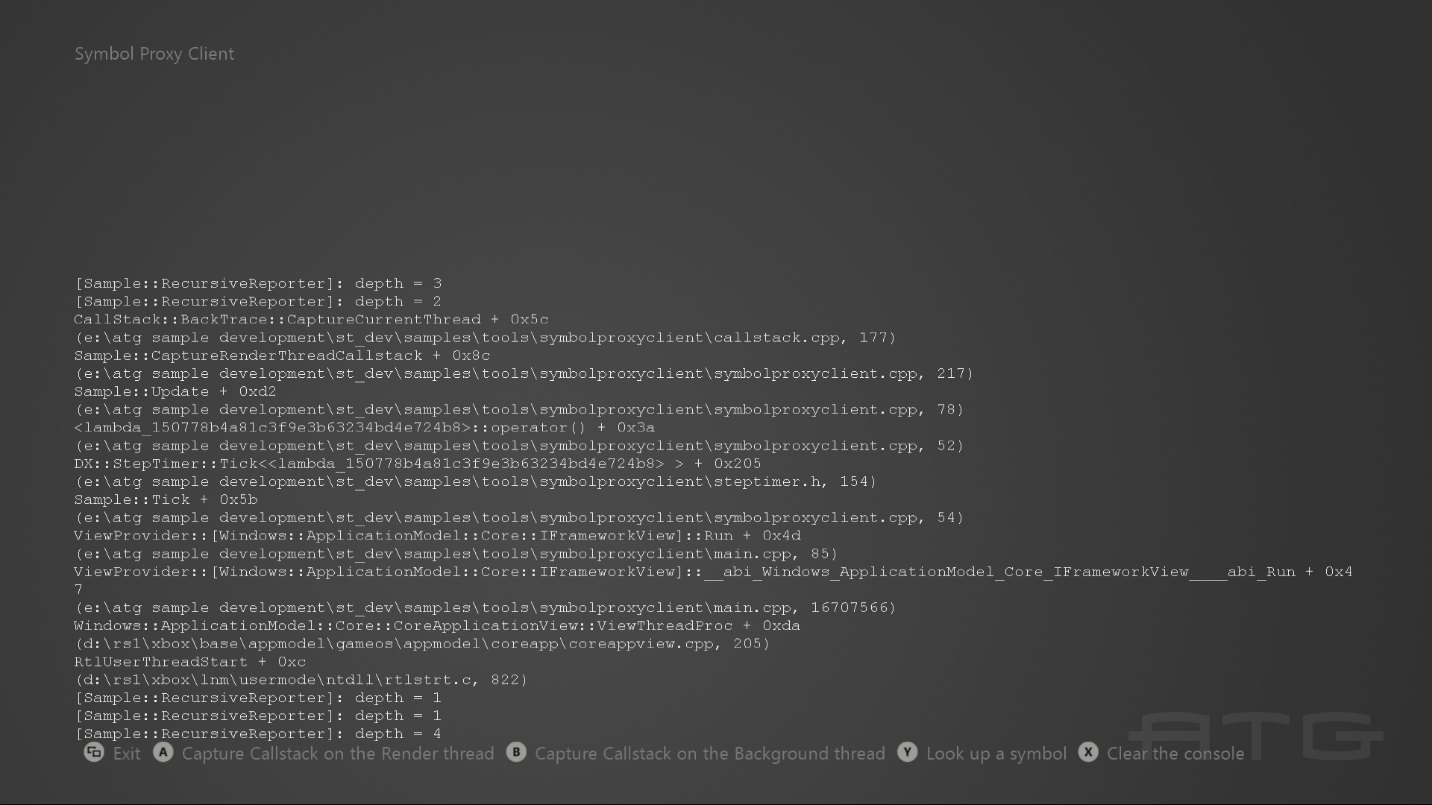
GetFrameCountFromContext()

CallStack.h/.cpp 中还有一个 BackTrace 类，简化了捕获反向跟踪和解析返回地址符号的过程。

# 使用样本

|  |  |
| --- | --- |
| 操作 | 游戏手柄 |
| 退出 | 查看按键 |
| 捕获渲染线程上的调用堆栈 | A 键 |
| 捕获后台线程上的调用堆栈 | B 键 |
| 查找符号 | Y 键 |
| 清除文本主机 | X 键 |

如需解析符号，必须在已配置为使用 xbConnect.exe 连接到主机的开发计算机上运行 xbSymbolProxy.exe。 如果不这样做，在无法解析任何符号时，样本将只会打印错误。



# 实施说明

**解析符号和性能**

为了解析符号，API 必须与桌面计算机上运行的 xbSymbolProxy.exe 进行通信。这是一个缓慢的操作过程，应在后台线程上异步执行，而不是在性能关键的线程上执行（例如，可以执行任何其他慢速操作。）为了演示此最佳实践，样本异步执行了所有符号解析。

BackTrace 类将捕获和解析步骤拆分为两种不同的操作。使用 BackTrace::CaptureCrossThread 和 BackTrace::CaptureCurrentThread 来捕获反向跟踪 - 这相当高效，而且在捕获反向跟踪的线程上无法避免性能成本。使用 BackTrace::Resolve() 与符号代理进行通信，并解析地址的符号信息。这可能非常慢，但是如果捕获了反向跟踪，则无需在同一线程上解析符号 - 因此可以安全地在后台线程上异步调用 BackTrace::Resolve()。

**暂停线程**

尝试捕获反向跟踪时，不应添加或删除堆栈帧。如果要捕获当前线程上的反向跟踪，这很容易实现。但是，如果要在不同的线程上捕获反向跟踪，则必须使用某种机制来“冻结”该线程。BackTrace 类在捕获反向跟踪之前和之后将间接调用 SuspendThread 和 ResumeThread。调用 SuspendThread 和 ResumeThread 将在名为 ThreadSuspender 的辅助类中进行管理。

线程暂停时，可能拥有某种同步基元，例如关键部分、互斥或信号灯。恢复暂停线程之前，如果暂停线程需要获取同步基元，这可能会造成死锁情况。这就是使用 SuspendThread 通常不是一个好主意的原因。

如果暂停线程需在另一个线程暂停时分配内存，则会发生特别有问题的情况。如果暂停线程碰巧在堆上持有锁，那么这将造成死锁。为了避免这种死锁情况，BackTrace 类使用全局静态缓冲区来存储反向跟踪的返回地址，从而避免在捕获反向跟踪时分配任何内存的需要。这一类为 64 堆栈帧留出了足够的空间。