

シンボル プロキシ クライアントのサンプル

*このサンプルには、2015 年 6 月以降の Xbox One XDK との互換性があります。*

*互換性に関する注意事項：xbSymbolProxy でソース行の情報が正しく解決されないという 6 月の XDK の既知の問題があります。これは 8 月の XDK で修正され、6 月の XDK QFE 1 に含まれる予定です*

# 概要

このサンプルは、SymbolResolve API を使用してコンソール上のアドレスのシンボル情報を解決する方法を紹介します。SymbolResolve.h には現在、GetSymbolFromAddress() と GetSourceLineFromAddress() の 2 つの API があります。これらの API は、たとえば、キャプチャしたバック トレースのリターン アドレスをデコードする場合など、コンソール側の診断コードに役立ちます。詳細については、XDK のドキュメントを参照してください。

このサンプルでは、コール スタックをキャプチャおよびデコードするための CallStack.h/.cpp にも便利なユーティリティ コードが用意されています。バック トレースをキャプチャするための 3 つのヘルパー関数：

現在のスレッドのバック トレースをキャプチャする。

CaptureBackTraceFromCurrentThread()

指定されたコンテキスト レコードから始まるバック トレースをキャプチャする。

CaptureBackTraceFromContext()

指定されたコンテキスト レコードから始まるコール スタック内のスタック フレーム数をカウントする。

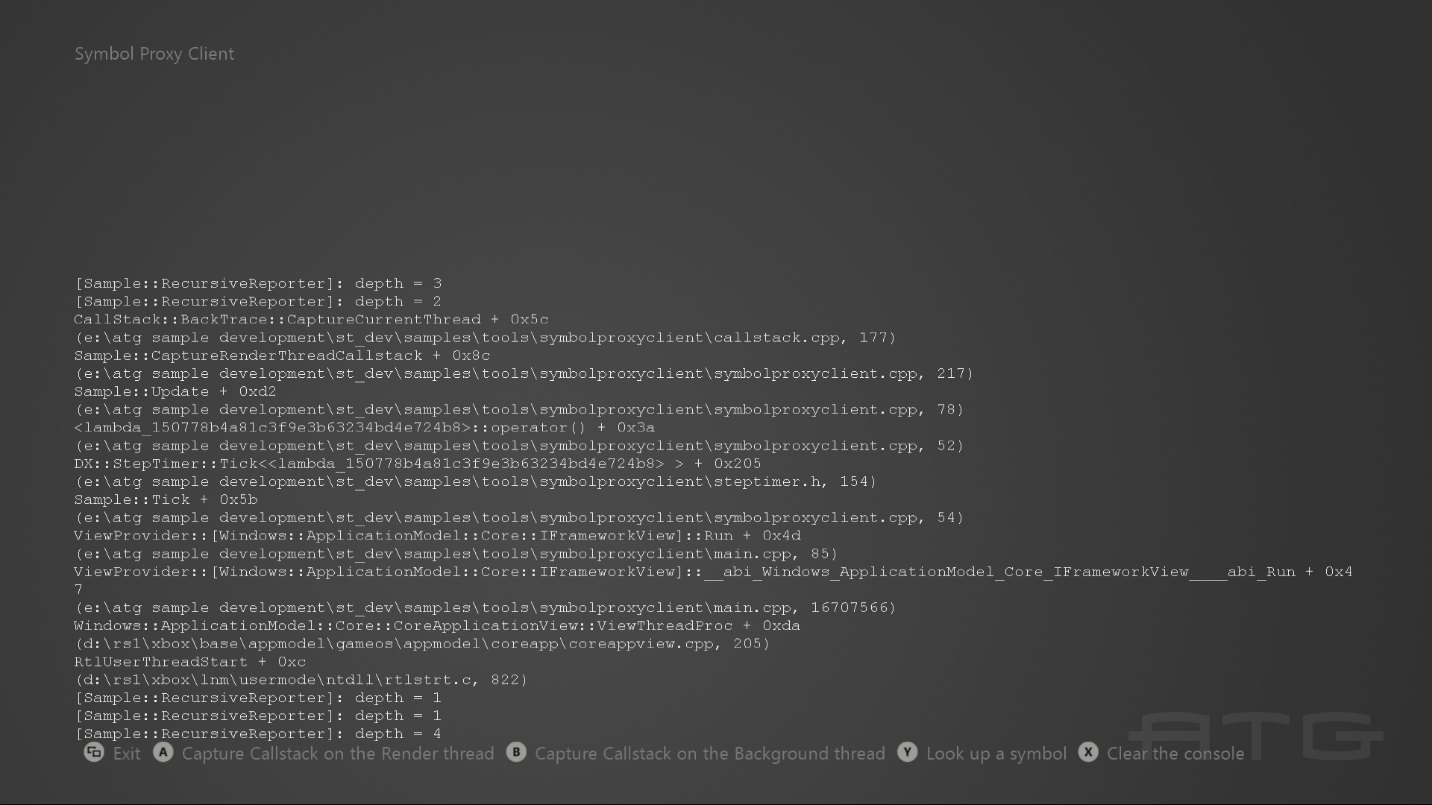
GetFrameCountFromContext()

また、CallStack.h/.cpp には BackTrace クラスがあり、バック トレースをキャプチャしてリターン アドレスのシンボルを解決するプロセスを簡素化します。

# サンプルの使用

|  |  |
| --- | --- |
| 作用 | ゲームパッド |
| 終了 | 表示ボタン |
| レンダー スレッドでコール スタックをキャプチャする | Aボタン |
| バックグラウンド スレッドでコール スタックをキャプチャする | B ボタン |
| シンボルを調べる | Y ボタン |
| テキスト コンソールを消去する | X ボタン |

シンボルを解決するには、xbConnect.exe を使用してコンソールに接続するように設定されている開発マシンで xbSymbolProxy.exe を実行する必要があります。 そうしないと、サンプルではシンボルを解決できないときにエラーが表示されるだけです。



# 実装上の注意

**シンボルとパフォーマンスの解決**

シンボルを解決するために、API はデスクトップ マシンで実行されている xbSymbolProxy.exe と通信する必要があります。これは低速の操作であり、パフォーマンス クリティカル スレッドではなく、バックグラウンド スレッドで非同期に実行する必要があります (他の低速の操作と一緒に行う場合など)。このベスト プラクティスを紹介するために、このサンプルではすべてのシンボル解決を非同期に実行します。

BackTrace クラスでは、キャプチャと解決の手順が 2 つの異なる操作に分割されます。バック トレースをキャプチャするには、BackTrace::CaptureCrossThread と BackTrace::CaptureCurrentThread を使用します。これは非常にパフォーマンスが高く、さらに、バック トレースをキャプチャしているスレッドではパフォーマンス コストが高くなってしまいます。シンボル プロキシと通信してアドレスのシンボル情報を解決するには、BackTrace::Resolve() を使用します。これは非常に低速になる可能性がありますが、バック トレースをキャプチャしたのと同じスレッド上のシンボルを解決する必要はありません。そのため、バックグラウンド スレッドで非同期に BackTrace::Resolve() を安全に呼び出すことができます。

**スレッドの中断**

バック トレースをキャプチャしようとしている間は、スタック フレームを追加したり、削除したりしないでください。現在のスレッドのバック トレースをキャプチャしている場合、これは簡単に実現できます。ただし、別のスレッドでバック トレースをキャプチャしている場合は、そのスレッドを「フリーズ」するために何らかのメカニズムを使用する必要があります。BackTrace クラスでは、バック トレースをキャプチャする前後に間接的に SuspendThread と ResumeThread を呼び出します。SuspendThread と ResumeThread の呼び出しは、ThreadSuspender という名前のヘルパー クラスで管理されます。

スレッドが中断された場合、ある種の同期プリミティブ (クリティカル セクション、ミューテックス、セマフォなど) が存在している可能性があることを理解することが重要です。中断しているスレッドが再開される前に、中断しているスレッドで同期化プリミティブの所有権を取得する必要がある場合、デッドロック シナリオが発生する可能性があります。このため、一般的に SuspendThread の使用は推奨されません。

中断しているスレッドで、他のスレッドが中断されている間にメモリを割り当てる必要がある場合、特に問題となるケースが 1 つ発生します。中断されたスレッドで、ヒープ上にロックが保持されていると、デッドロックが発生します。このようなデッドロックのシナリオを回避するために、BackTrace クラスではバック トレースのリターン アドレスを格納するためにグローバルな静的バッファーを使用するため、バック トレースのキャプチャ中にメモリを割り当てる必要がなくなります。このクラスでは、64 ビット モードのスタック フレーム用に十分なスペースが確保されます。