SOS & ICorDebug

# Introduction

The purpose of this doc is to capture the work and learnings produced by the Dev 11 MQ item to integrate the public ICorDebug\* interfaces (mscordbi.dll) into the SOS implementation.

Note: Use of the name “windbg” in this document is meant to actually indicate all members of the ntsd family of debuggers.

# MSCORDBI.DLL Distribution

## What’s checked in - suboptimal

For MQ, SOS activates mscordbi.dll via the “typical” mechanism of calling into mscoree.dll to get the ICLRDebugging interface, on which SOS calls OpenVirtualProcess. Although this works, it sucks. It assumes the CLR you want to debug has been formally installed onto the debugging machine (complete with mscoree.dll in your system32 / syswow64 directory).

## Recommendation

Optimally, everything windbg needs in order to debug the CLR should be automagically downloaded to the user’s box, including DAC, mscordbi.dll, and even SOS itself, with no need at all for mscoree.dll or mscoreei.dll.

### Download

Today, DAC is already automagically downloaded via the symbol server, with support for that hard-coded into the debugger: Windbg takes file information of the CLR.dll in the debuggee or dump that would normally be used to ask the symbol server for CLR.pdb, and instead asks for the DAC dll (using CLR.dll’s file *info* and the DAC dll’s file *name*). This same hack could be applied to download the appropriate mscordbi.dll, and even SOS.dll.

This requires work from the following parties:

* **CLR & Windows debugging teams** agree on a design whereby windbg could automatically (or receive a command to) locate and download the appropriate mscordbi.dll and SOS.dll much as it does today to locate DAC. Extensions like SOS should then be able to access mscordbi.dll via a windbg API, much like the windbg API that exposes the private DAC interface.
  + The above is intentionally way-vague. Extra work needs to be done to determine a good way for windbg & extensions to cooperate and access mscordbi functionality.
* **Windows debugging team** implements the above
* **CLR Servicing Team** must alter their process to ensure mscordbi.dll and SOS.dll are properly indexed on the symbol server upon each CLR release, just as with DAC.

### Activation

Once DAC, SOS, and mscordbi are all on the same machine, we need a way for SOS to get an ICorDebugProcess from mscordbi that does not require mscoree.dll on the box. Since the necessary mscordbi is already available as per above, there is no need for SOS to go through mscoree or mscoreei as it is today. Since mscoreei.dll eventually just calls OpenVirtualProcessImpl() from mscordbi.dll, one could imagine that SOS could simply call mscordbi’s OpenVirtualProcessImpl() directly.

* **CLR Team** would need to export OpenVirtualProcessImpl, or the equivalent, from mscordbi.dll directly.
* **CLR Team** would modify SOS to use OpenVirtualProcessImpl rather than the current scheme that relies on mscoree.dll.
* Since CLR Team owns both ends of this interface (i.e., implementation and consumer of OpenVirtualProcessImpl), and since we already have a scheme to exchange version numbers in OpenVirtualProcessImpl, there should not be versioning problems.
* However, we might prefer to have windbg (instead of SOS) call OpenVirtualProcessImpl(). Examples why:
  + The “Download” section above talks about SOS “accessing” mscordbi. This is vague, and could either mean that SOS calls into mscordbi’ s OpenVirtualProcessImpl() directly to get an ICorDebugProcess OR SOS uses a new, structured windbg API which would call (on SOS’s behalf) into mscordbi to provide an ICorDebugProcess. The latter may be a more natural way for an extension to get access to the “right” ICorDebugProcess, particularly if multiple mscordbi.dll’s end up on the symbol path, or for in-proc SxS scenarios.
  + Perhaps windbg could someday use mscordbi directly to aid in managed debugging, rather than the private DAC interface.

If we do choose to have windbg (instead of SOS) call OpenVirtualProcessImpl, then we’d need to give some more thought to OpenVirtualProcessImpl and versioning, since we no longer control the consumer end of that relationship.

# !ClrStack –i [-p]

This checkin uses !clrstack as the prototype for activating and consuming the public dbgapi. This section briefly describes how that works.

If “-i” is specified, then !clrstack defers to ClrStackFromPublicInterface() to do its work. Today in SOS, the private DAC interface is initialized at the beginning of each command, and released at the end of each command. ClrStackFromPublicInterface() follows suit by initializing the public interface at the top, via ICLRDebugging, thus setting g\_pCorDebugProcess appropriately (see “What’s checked in – suboptimal” above). *Note: currently, g\_pCorDebugProcess is not released but hey, prototype code.*

ClrStackFromPublicInterface then uses g\_pCorDebugProcess to grab the other public interfaces necessary to do a stackwalk. If “-p” is specified, ClrStackFromPublicInterface will also use ICorDebug\* to get parameter info.

## Caveats

To pretty-print the function names, there are several metadata-consuming chunks of code in SOS. I picked the one that required the least amount of private DAC gunk. However, that code appears not to be well-exercised. See comments in the code for more details.

The code in !clrstack –i to print the stack is only mildly tested. There may well be cases where the stack doesn’t look so great or where managed frames appear unordered with respect to the internal (explicit EE) Frames. See comments in the code for more details.

# Mac

Nope. There are several silly reasons why the above is #ifdef’d out on the Mac. The work necessary to enable this on the Mac may well be straightforward. Some of the reasons the Mac compile failed for me:

* Need psapi.h for GetModuleInformation and structures it uses. The equivalents need to be found (or implemented!) for the Mac.
* No IID defined for IDebugAdvanced3. (Possibly just a matter of manually rebuilding the right GUID lib for the Mac, if you can find it.)
* Couldn’t find some CLR-specific headers on the Mac build, like **metahost.h**.