The Quest For A Better Crash

Victor Ciura

C++ now

2021
MAY 2-7
Aspen, Colorado, USA

The Quest For A Better Crash

C++ Now 2021

May 7


@ciura_victor

Victor Ciura
Principal Engineer


CAPHYON

Abstract

Crashed ! Now What ? “It works on my machine” :) Those little words that make the heart of QAs and clients skip a beat. Sometimes reproducing a crash on a developer's machine is next to impossible. Most of the time remote debugging is out of the question and all you’re left with are some scant log files and maybe a memory dump file, if you’re lucky. Wouldn’t you like to know the exact point of failure in the program and how it got there, on the client’s PC ?

How can you get your hands on a StackTrace of that crash on the client’s machine ? And how can you make any sense of it without symbols (client deployed Release build) ?
In this session, I’ll present a Windows specific technique we developed, that my team uses regularly to debug such scenarios in production. We leverage OS APIs like the Image Help Library (ImageHlp.dll), the Debug Help Library (DbgHelp.dll) to work with PE/COFF images and PDBs and reconstruct symbolicated StackTraces for Release crashes in production. The technique and APIs work all the way from Windows XP up to Windows 10, both for x86 and x64 executables.

We’ll see how symbols are loaded and how PDBs work, we’ll discuss partial/incremental PDBs and we’ll have to get comfortable with Structured Exception Handling (SEH). Did I mention Address Space Layout Randomization (ASLR) ? This is going to be fun :)

Come with me on this journey and we’ll walk the stack together, to reconstruct each frame, from a few pointers and some symbols.

From highly efficient platform-specific implementations, to boost::Stacktrace, to P0881, to C++23... we’ll analyze together the requirements, constraints and advantages of each design decision.

Online conference

Due to the nature of delivery medium &
streaming delays, I prefer to take questions at the end.

Q & A

About me

Advanced Installer

Clang Power Tools

@ciura_victor

Vignette in 3 parts

Remember the crash

Roll your own

The Future: post-pandemic crashes

Disclaimer

Windows*

x86/x64

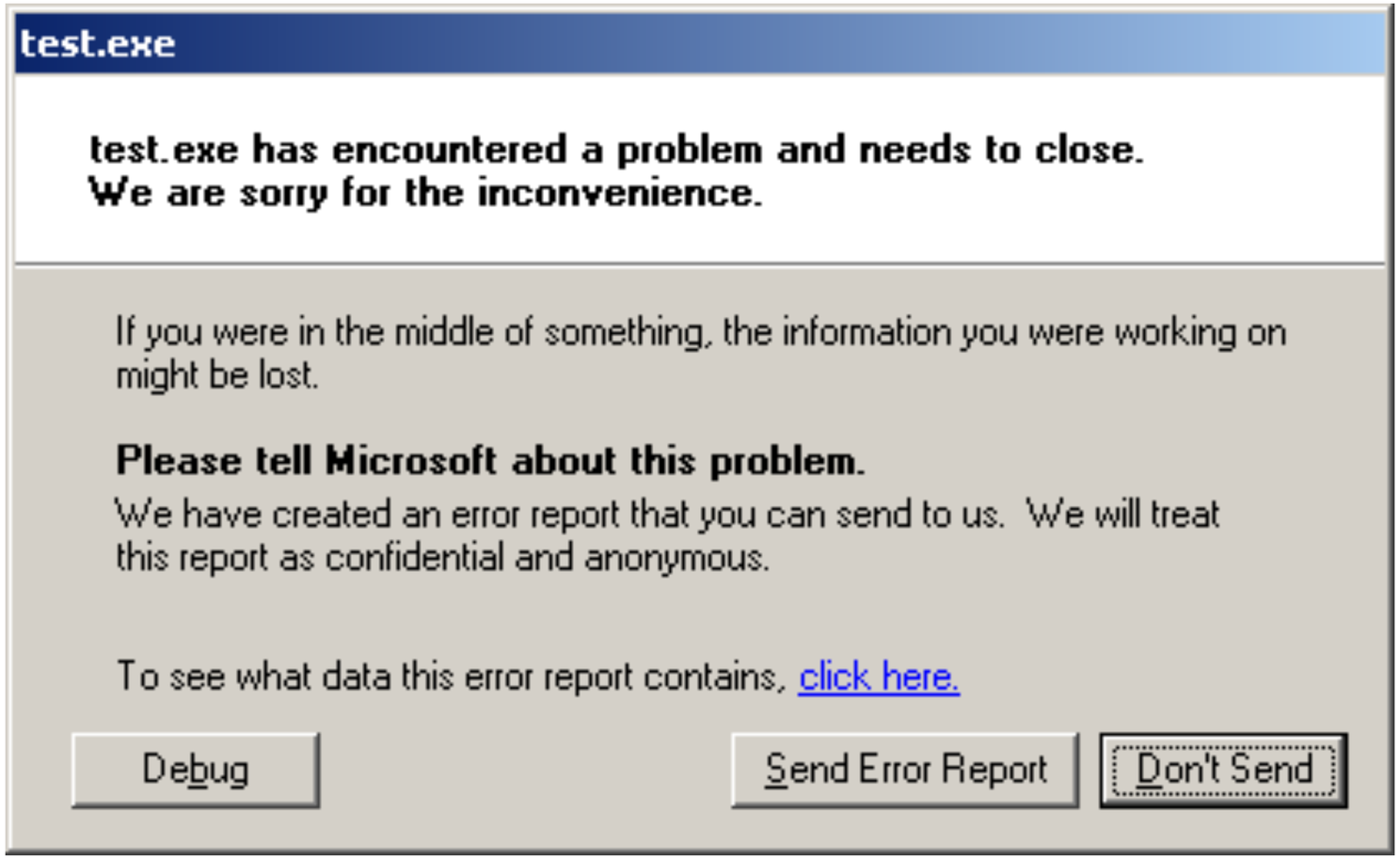
* Mostly, with some bits about ISO C++ (P0881) at the end

Part I

Remember the crash

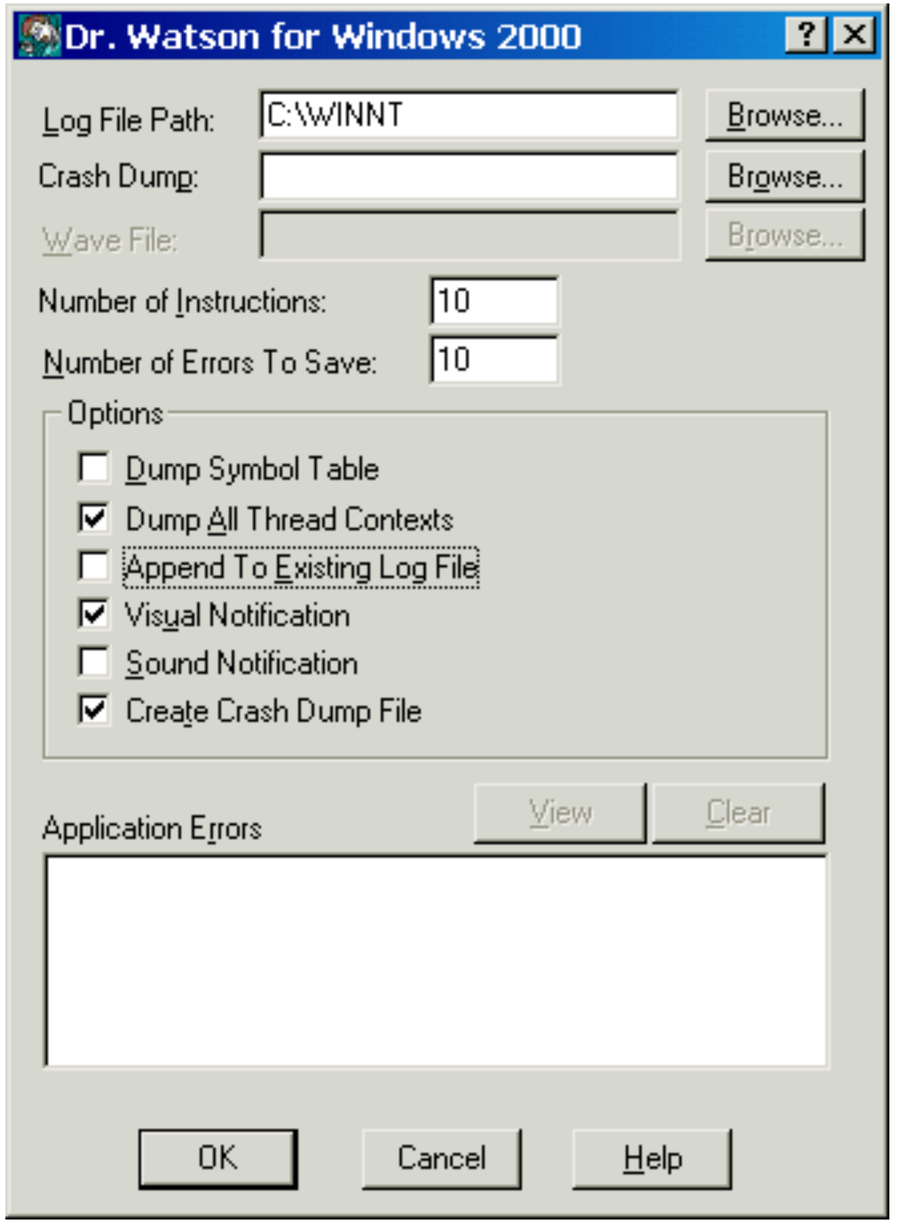
Remember the crash

We've all been there...



Remember the crash

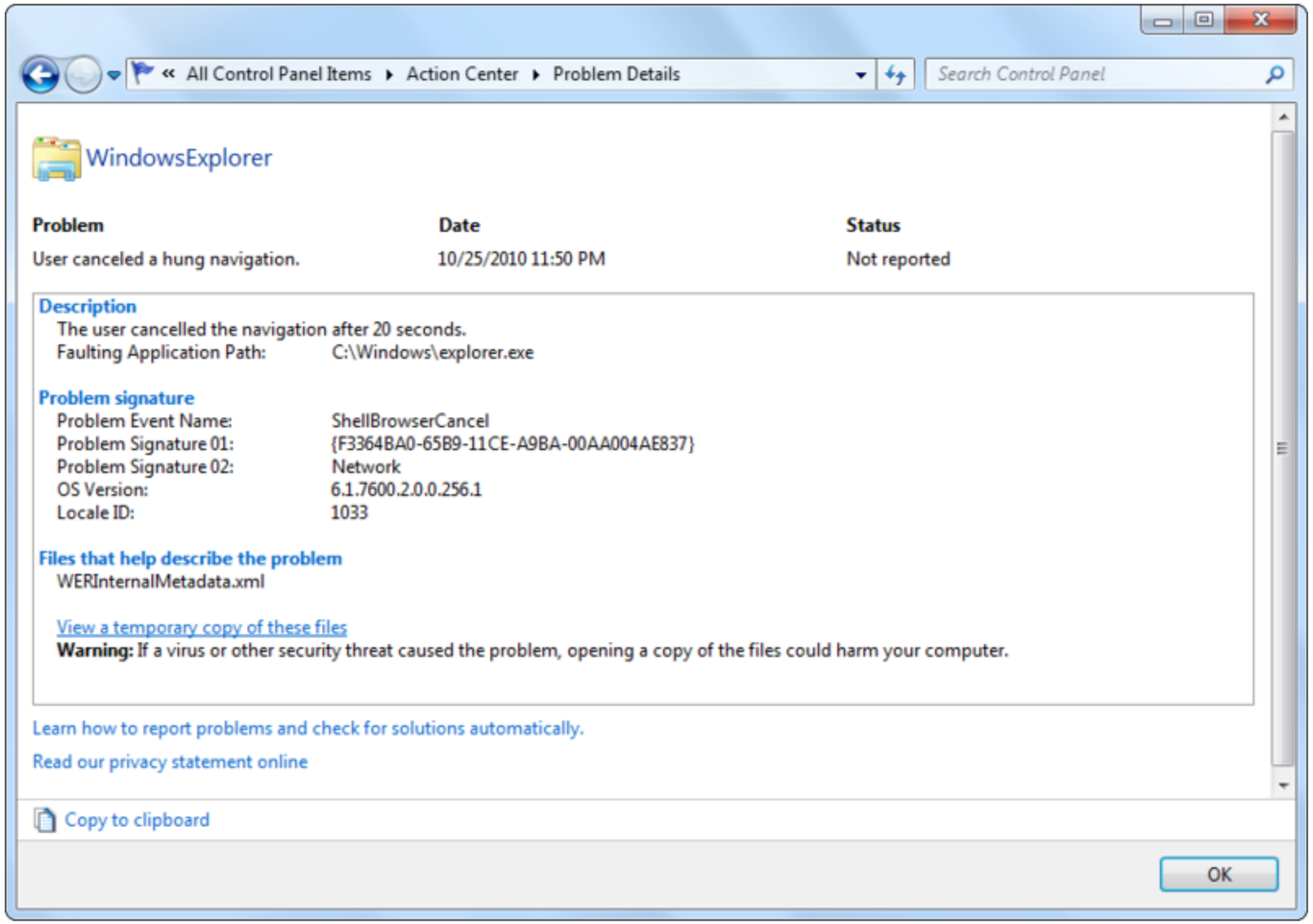
Some of us might even remember our old friend Dr. Watson



Remember the crash

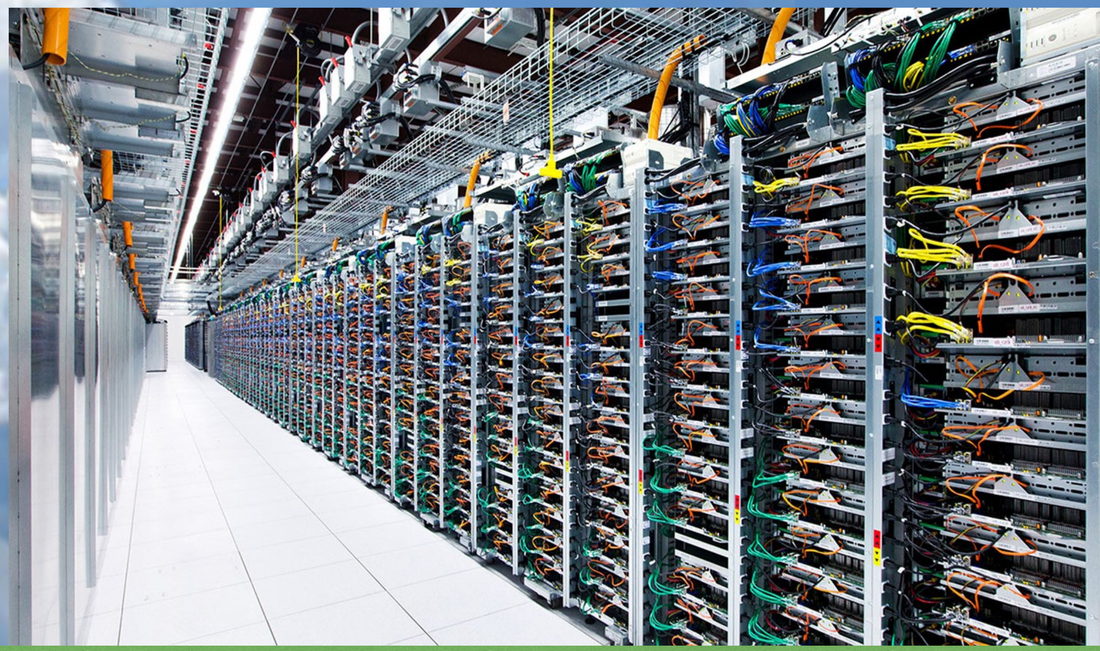
... or his modern friend,

Windows Error Reporting



Crash Paradise

But, where do all these crashes go?



Crash Paradise

Questions I had over 10 years ago,
before any Store or Windows Dev Center

Crash Paradise

Is this even available for third-party Windows apps ?

Questions I had over 10 years ago,
before any Store or Windows Dev Center

Crash Paradise

Is this even available for third-party Windows apps ?

How can we register to receive such crash dumps ?

Questions I had over 10 years ago,
before any Store or Windows Dev Center

Crash Paradise

Is this even available for third-party Windows apps ?

How can we register to receive such crash dumps ?

What does it cost ?

Questions I had over 10 years ago,
before any Store or Windows Dev Center

Crash Paradise

Is this even available for third-party Windows apps ?

How can we register to receive such crash dumps ?

What does it cost ?

How does the crash data look like ? Who owns it ?

Questions I had over 10 years ago,
before any Store or Windows Dev Center

Stack Trace

I just wanted this...

```
#0 0x793d62f6 in __asan_wrap_memset d:\_work\5\s\llvm\projects\compiler-rt\lib\sanitizer_common\sanitizer_common_interceptors.inc:764
#1 0x77dd46e7  (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2c46e7)
#2 0x77dd4ce1  (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2c4ce1)
#3 0x75d408fe  (C:\WINDOWS\System32\KERNELBASE.dll+0x100f08fe)
#4 0xa5ada0 in try_get_first_available_module minkernel\crts\ucrt\src\appcrt\internal\winapi_thunks.cpp:271
#5 0xa5ae99 in try_get_function minkernel\crts\ucrt\src\appcrt\internal\winapi_thunks.cpp:326
#6 0xa5b028 in __acrt_AppPolicyGetProcessTerminationMethodInternal minkernel\crts\ucrt\src\appcrt\internal\winapi_thunks.cpp:737
#7 0xa606ad in __acrt_get_process_end_policy minkernel\crts\ucrt\src\appcrt\internal\win_policies.cpp:84
#8 0xa52dcb in exit_or_terminate_process minkernel\crts\ucrt\src\appcrt\startup\exit.cpp:134
#9 0xa52da7 in common_exit minkernel\crts\ucrt\src\appcrt\startup\exit.cpp:280
#10 0xa52fb6 in exit minkernel\crts\ucrt\src\appcrt\startup\exit.cpp:293
#11 0xa2deb3 in _scrt_common_main_seh d:\agent\_work\2\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:295
#12 0x75ef6358  (C:\WINDOWS\System32\KERNEL32.DLL+0x6b816358)
#13 0x77df7a93  (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2e7a93)
```

Crash Paradise

The answers turned out to be... *complicated* :(

docs.microsoft.com/en-us/windows/win32/debug/symbol-servers-and-symbol-stores

Crash Paradise

The answers turned out to be... *complicated* :(

... custom registration for each app version,
Microsoft **Symbol Server** instances (deployed on-premise),
Symbol Stores, etc.

... a DevOps nightmare!

docs.microsoft.com/en-us/windows/win32/debug/symbol-servers-and-symbol-stores

Part II

Roll your own

Crash Love

Like any good programmer, I decided to build my own 

Crash Love

Like any good programmer, I decided to build my own 

Goals:

Crash Love

Like any good programmer, I decided to build my own 

Goals:

Crash Love

Like any good programmer, I decided to build my own 

Goals:

- quick to develop

Crash Love

Like any good programmer, I decided to build my own 

Goals:

- quick to develop
- easy to integrate into our CI/CD (no special service, symbol server)

Crash Love

Like any good programmer, I decided to build my own 

Goals:

- quick to develop
- easy to integrate into our CI/CD (no special service, symbol server)
- zero footprint on client side (not shipping symbols)

Crash Love

Like any good programmer, I decided to build my own 

Goals:

- quick to develop
- easy to integrate into our CI/CD (no special service, symbol server)
- zero footprint on client side (not shipping symbols)
- zero perf impact on Release binaries (on the happy path)

Crash Love

Like any good programmer, I decided to build my own 

Goals:

- quick to develop
- easy to integrate into our CI/CD (no special service, symbol server)
- zero footprint on client side (not shipping symbols)
- zero perf impact on Release binaries (on the happy path)
- easy to use standalone tool (non-dev machine) for processing crash reports

Workflow

CI/CD Pipelines

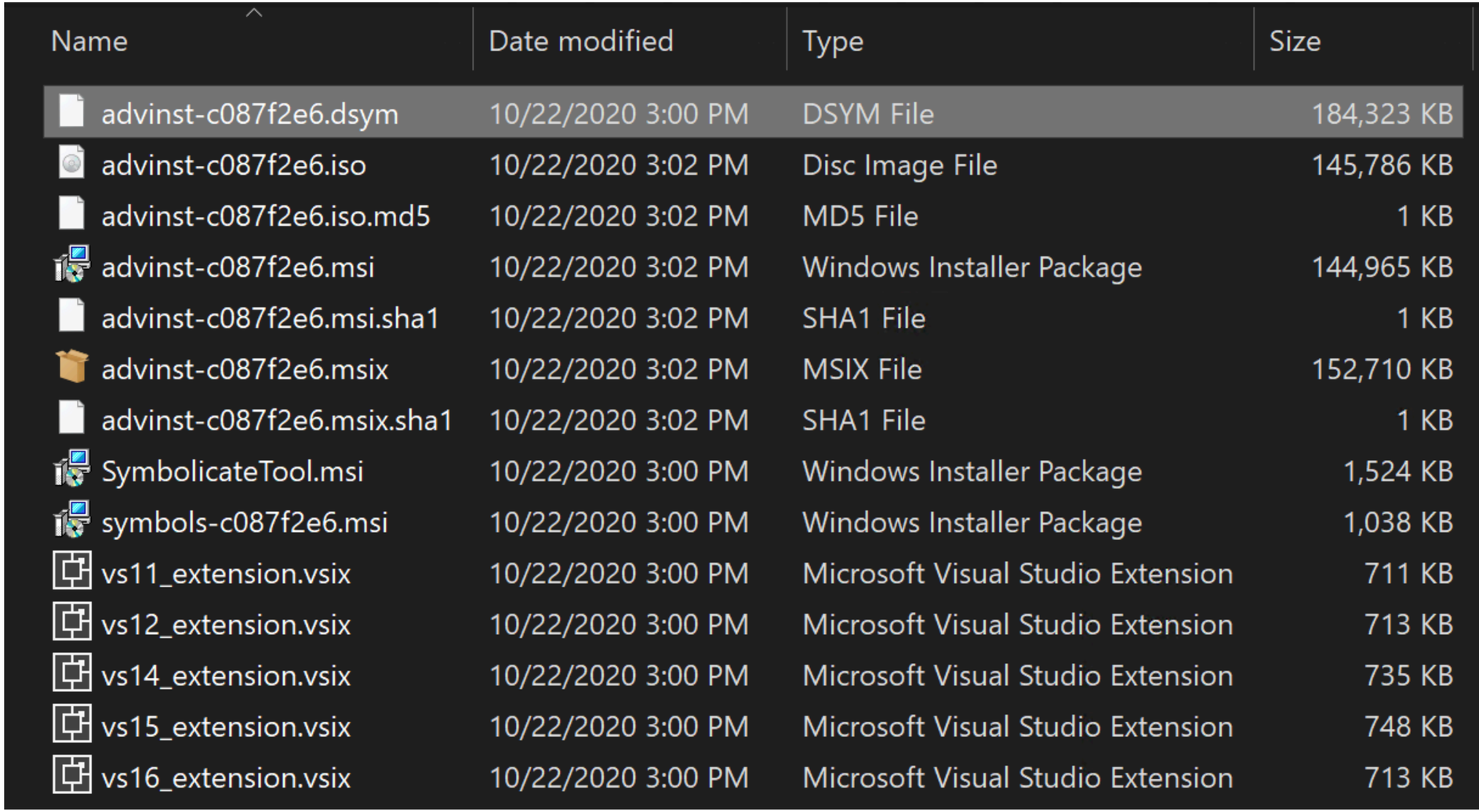
Jenkins **GitLab** **Build artifacts**

CI/CD Artifacts

Build artifacts for git commit: `c087f2e6`

symbols archive 👉

standalone tool for processing crash reports {



Name	Date modified	Type	Size
advinst-c087f2e6.dsym	10/22/2020 3:00 PM	DSYM File	184,323 KB
advinst-c087f2e6.iso	10/22/2020 3:02 PM	Disc Image File	145,786 KB
advinst-c087f2e6.iso.md5	10/22/2020 3:02 PM	MD5 File	1 KB
advinst-c087f2e6.msi	10/22/2020 3:02 PM	Windows Installer Package	144,965 KB
advinst-c087f2e6.msi.sha1	10/22/2020 3:02 PM	SHA1 File	1 KB
advinst-c087f2e6.msix	10/22/2020 3:02 PM	MSIX File	152,710 KB
advinst-c087f2e6.msix.sha1	10/22/2020 3:02 PM	SHA1 File	1 KB
SymbolicateTool.msi	10/22/2020 3:00 PM	Windows Installer Package	1,524 KB
symbols-c087f2e6.msi	10/22/2020 3:00 PM	Windows Installer Package	1,038 KB
vs11_extension.vsix	10/22/2020 3:00 PM	Microsoft Visual Studio Extension	711 KB
vs12_extension.vsix	10/22/2020 3:00 PM	Microsoft Visual Studio Extension	713 KB
vs14_extension.vsix	10/22/2020 3:00 PM	Microsoft Visual Studio Extension	735 KB
vs15_extension.vsix	10/22/2020 3:00 PM	Microsoft Visual Studio Extension	748 KB
vs16_extension.vsix	10/22/2020 3:00 PM	Microsoft Visual Studio Extension	713 KB

CI/CD Artifacts

Symbols archive for a **Release** build

C:\Users\victo\Downloads\advinst-c087f2e6.dsym\advinst-c087f2e6\bin\x86\

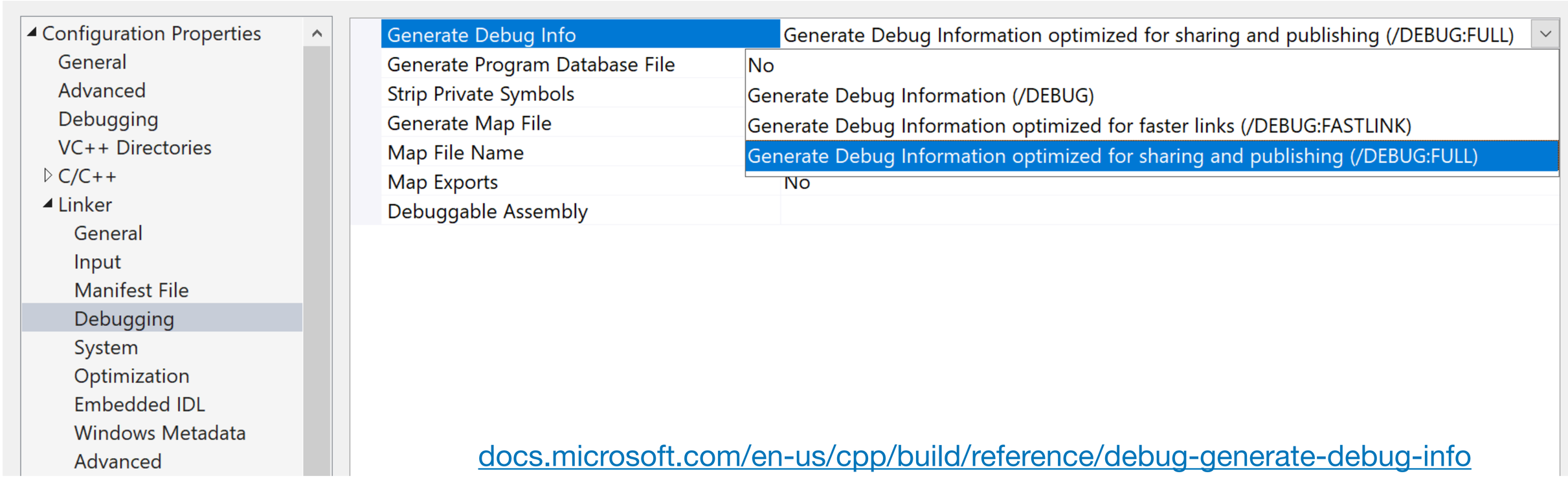
Name	Size	Packed Size	Modified	Created	Access...	Attribu...	Encryp...	Comm...	CRC	Method
..										
advinst.pdb	506 286 080	118 968 797	2020-10-22 13:46			A	-		4DE785A5	Deflate
ExternalUi.pdb	27 258 880	6 865 327	2020-10-22 13:47			A	-		2D5D16DC	Deflate
Repackager.pdb	74 829 824	16 813 693	2020-10-22 13:50			A	-		8891255E	Deflate
VmLauncher.pdb	45 469 696	10 516 761	2020-10-22 13:53			A	-		09A02E1B	Deflate

`advinst-c087f2e6.dsym` - basically, a ZIP bundle

Symbols

PDB == 42

Use `/DEBUG:FULL` for Release builds & archiving symbols



docs.microsoft.com/en-us/cpp/build/reference/debug-generate-debug-info

No Symbols

You don't want to see a dry stack trace (crash report)

```
[SEH_AV_WRITE_NULLPTR] ACCESS_VIOLATION (0xc0000005) at address [0x000000014002772f]

Advanced Repackager (x64) 12.8 build 69285
*** Stack Trace (x64) ***

[0x000000014002772f]  -----
[0x000000014002911c]  -----
[0x0000000140028f66]  -----
[0x0000000140026f86]  -----
[0x0000000140026e68]  -----
[0x0000000140020be8]  -----
[0x0000000076b979b7]  CreateDialogParamW()
[0x0000000076b97792]  CreateDialogParamW()
[0x0000000076b976c2]  CreateDialogParamW()
[0x0000000076b89bd1]  TranslateMessageEx()
[0x0000000076b86aa8]  SetTimer()
[0x0000000076b86bad]  SendMessageW()
[0x000007fefc0092a0]  Ordinal342()
[0x000007fefc008604]  Ordinal342()
[0x000007fefc0217bd]  -----
[0x000007fefc023075]  -----
[0x000007fefc023223]  -----
[0x000007fefc024491]  -----
[0x0000000076b979b7]  -----
[0x0000000076b97792]  -----
[0x0000000076b976c2]  -----
[0x0000000076b89bd1]  TranslateMessageEx()
[0x0000000076b83bfc]  CallWindowProcW()
[0x0000000076b83b78]  CallWindowProcW()
[0x000000014003268e]  -----
[0x0000000140031f33]  -----
[0x0000000140032257]  -----
[0x00000001400312d4]  -----
```

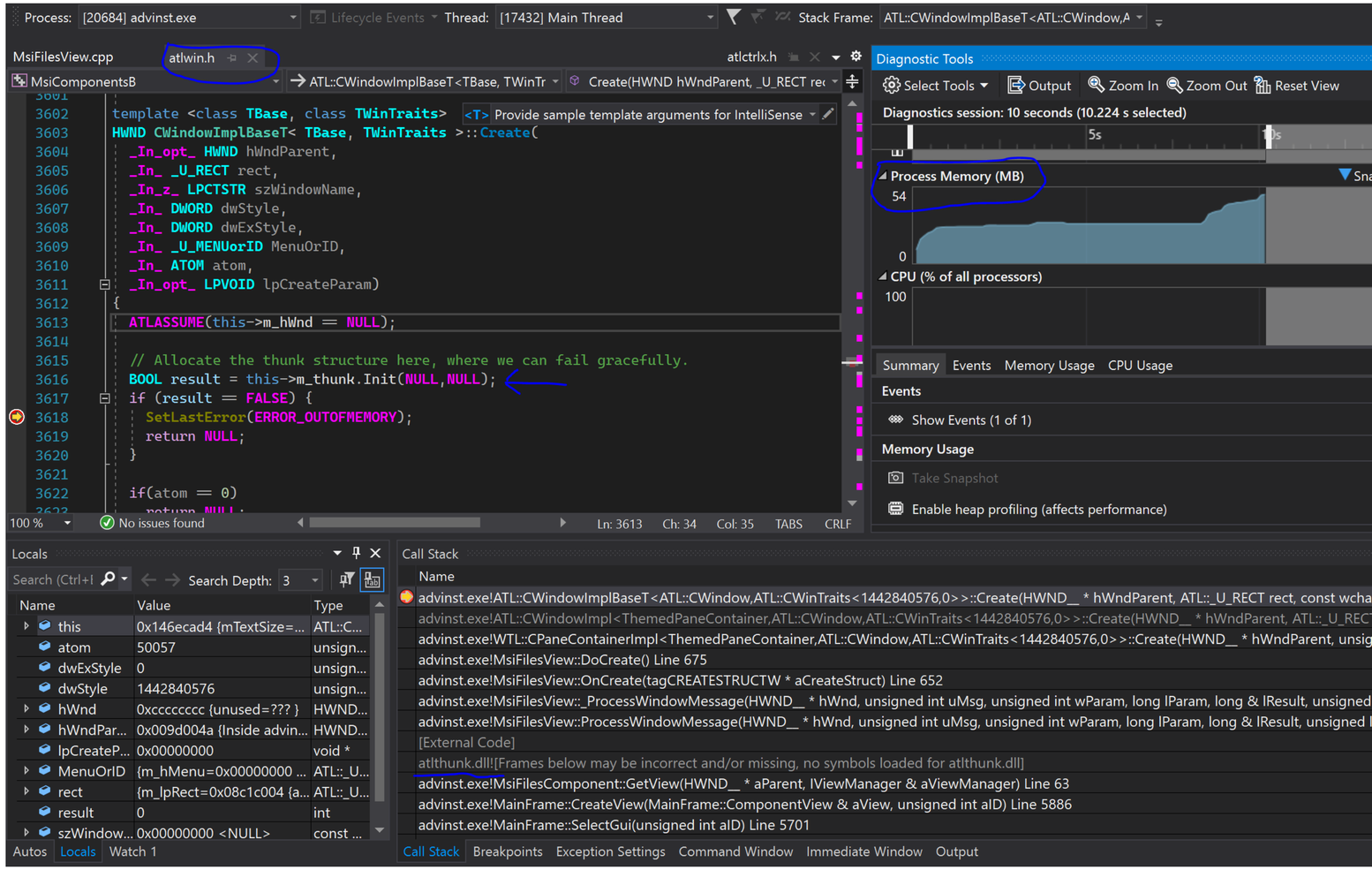
Stack Trace Symbols

Victor Zverovich (vitaut)
@vzverovich

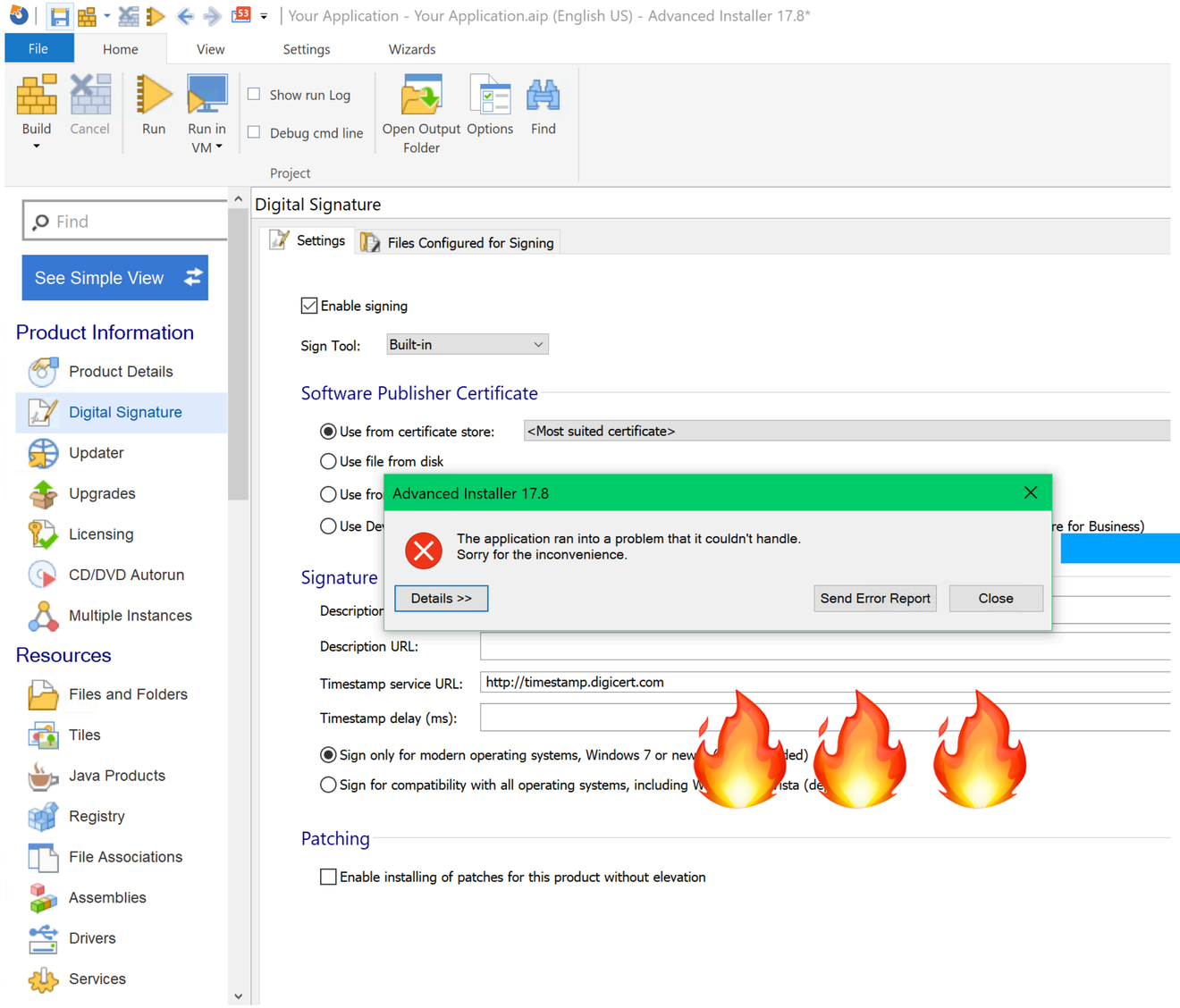
When someone asks if a stack trace looks familiar...

Context

TFW you've got a fresh repro case and you can dive into a debugging session...



Sometimes you crash



crash report
(no symbols)

Support service

- collect
- symbolicate
- triage

Symbolicate Tool

Symbolicate Tool (x64)

Extract Symbol Package

MSI Package: C:\Users\victo\Downloads\advinst17.6\advinst-c087f2e6.msi ...

Loaded Symbols: bin\x64\Repackager

Debugee

EXE File: C:\Users\victo\AppData\Local\Temp\Symbol Packages\advinst-c087f2e6\bin\x64\Repackager.exe ...

PDB File: C:\Users\victo\AppData\Local\Temp\Symbol Packages\advinst-c087f2e6\bin\x64\Repackager.pdb ...

Command Line:

Stack Trace

Dump File: C:\Users\victo\Downloads\Crash Report.st ...

```
[SEH_AV_WRITE_NULLPTR] ACCESS_VIOLATION (0xc0000005) at address [0x000000014002772f]

Advanced Repackager (x64) 17.6 build c087f2e6
*** Stack Trace (x64) ***

[0x000000014002772f]  ProductDetailsPage::OnWizardNext() -> productdetailspage.cpp:97
[0x000000014002911c]  WTL::CPropertyPageImpl<ProductDetailsPage,WTL::CPropertyPageWind
[0x0000000140028f66]  CResizablePropertyPageImpl<ProductDetailsPage,WTL::CPropertyPage
[0x0000000140026f86]  ProductDetailsPage::_ProcessWindowMessage() -> productdetailspag
[0x0000000140026e68]  ProductDetailsPage::ProcessWindowMessage() -> productdetailspage
[0x0000000140020be8]  ATL::CDialogImplBaseT<WTL::CPropertyPageWindow>::DialogProc() ->
[0x0000000076b979b7]  CreateDialogParamW()
[0x0000000076b97792]  CreateDialogParamW()
[0x0000000076b976c2]  CreateDialogParamW()
[0x0000000076b89bd1]  TranslateMessageEx()
[0x0000000076b86aa8]  SetTimer()
[0x0000000076b86bad]  SendMessageW()
[0x000007fefc0092a0]  Ordinal342()
[0x000007fefc008604]  Ordinal342()
[0x000007fefc0217bd]  CreatePropertySheetPage()
```

Symbolicate !

Symbolicate Tool

Symbolicate Tool (x64)

Extract Symbol Package

MSI Package: C:\Users\victo\Downloads\advinst17.6\advinst-c087f2e6.msi ...

Loaded Symbols: bin\x64\Repackager

Debugee

EXE File: C:\Users\victo\AppData\Local\Temp\Symbol Packages\advinst-c087f2e6\bin\x64\Repackager.exe ...

PDB File: C:\Users\victo\AppData\Local\Temp\Symbol Packages\advinst-c087f2e6\bin\x64\Repackager.pdb ...

Command Line:

Stack Trace

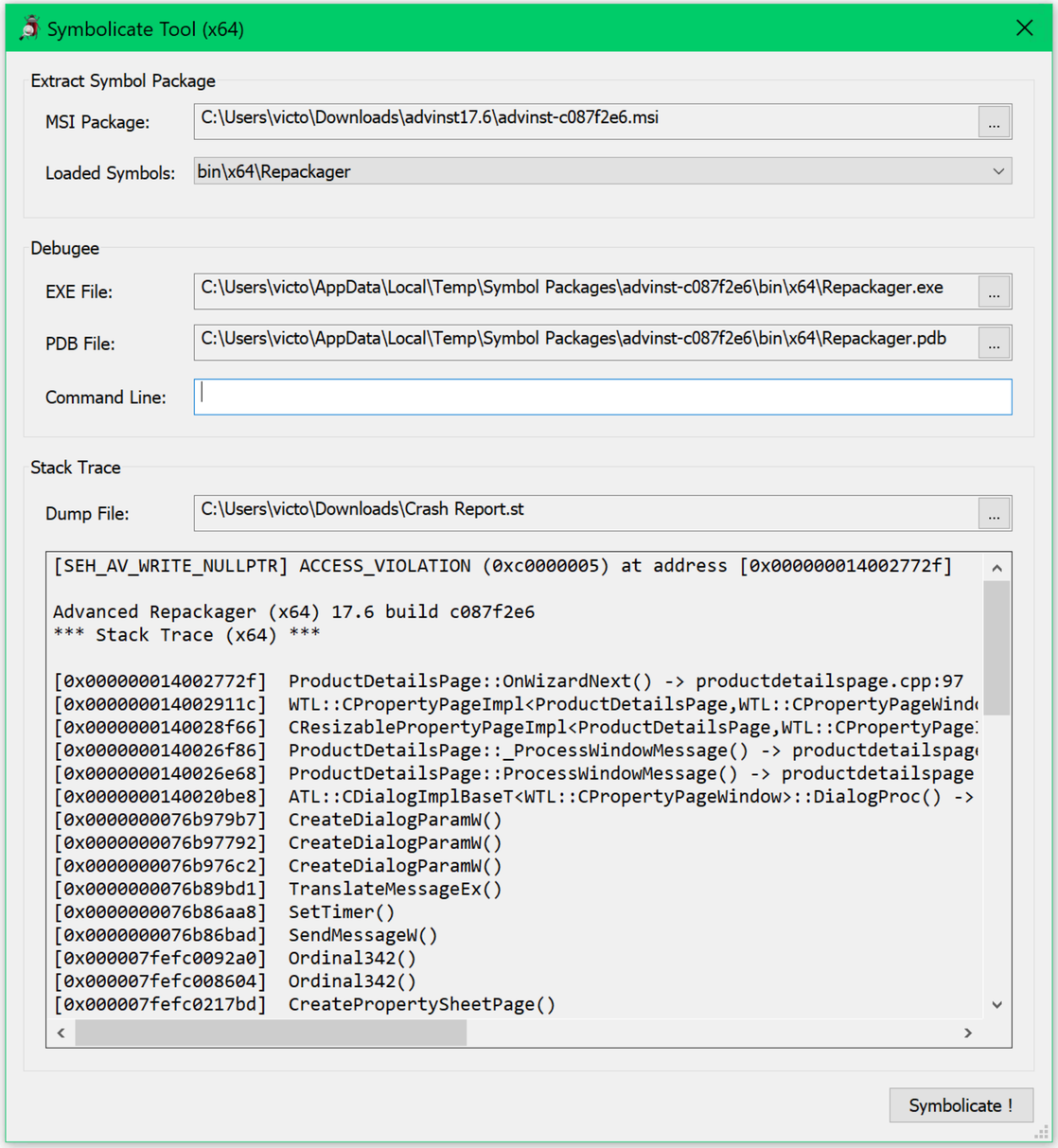
Dump File: C:\Users\victo\Downloads\Crash Report.st ...

```
[SEH_AV_WRITE_NULLPTR] ACCESS_VIOLATION (0xc0000005) at address [0x000000014002772f]

Advanced Repackager (x64) 17.6 build c087f2e6
*** Stack Trace (x64) ***

[0x000000014002772f]  ProductDetailsPage::OnWizardNext() -> productdetailspage.cpp:97
[0x000000014002911c]  WTL::CPropertyPageImpl<ProductDetailsPage,WTL::CPropertyPageWind
[0x0000000140028f66]  CResizablePropertyPageImpl<ProductDetailsPage,WTL::CPropertyPage
[0x0000000140026f86]  ProductDetailsPage::_ProcessWindowMessage() -> productdetailspag
[0x0000000140026e68]  ProductDetailsPage::ProcessWindowMessage() -> productdetailspage
[0x0000000140020be8]  ATL::CDialogImplBaseT<WTL::CPropertyPageWindow>::DialogProc() ->
[0x0000000076b979b7]  CreateDialogParamW()
[0x0000000076b97792]  CreateDialogParamW()
[0x0000000076b976c2]  CreateDialogParamW()
[0x0000000076b89bd1]  TranslateMessageEx()
[0x0000000076b86aa8]  SetTimer()
[0x0000000076b86bad]  SendMessageW()
[0x000007fefc0092a0]  Ordinal342()
[0x000007fefc008604]  Ordinal342()
[0x000007fefc0217bd]  CreatePropertySheetPage()
```

Symbolicate !

⬅️ select build

Symbolicate Tool

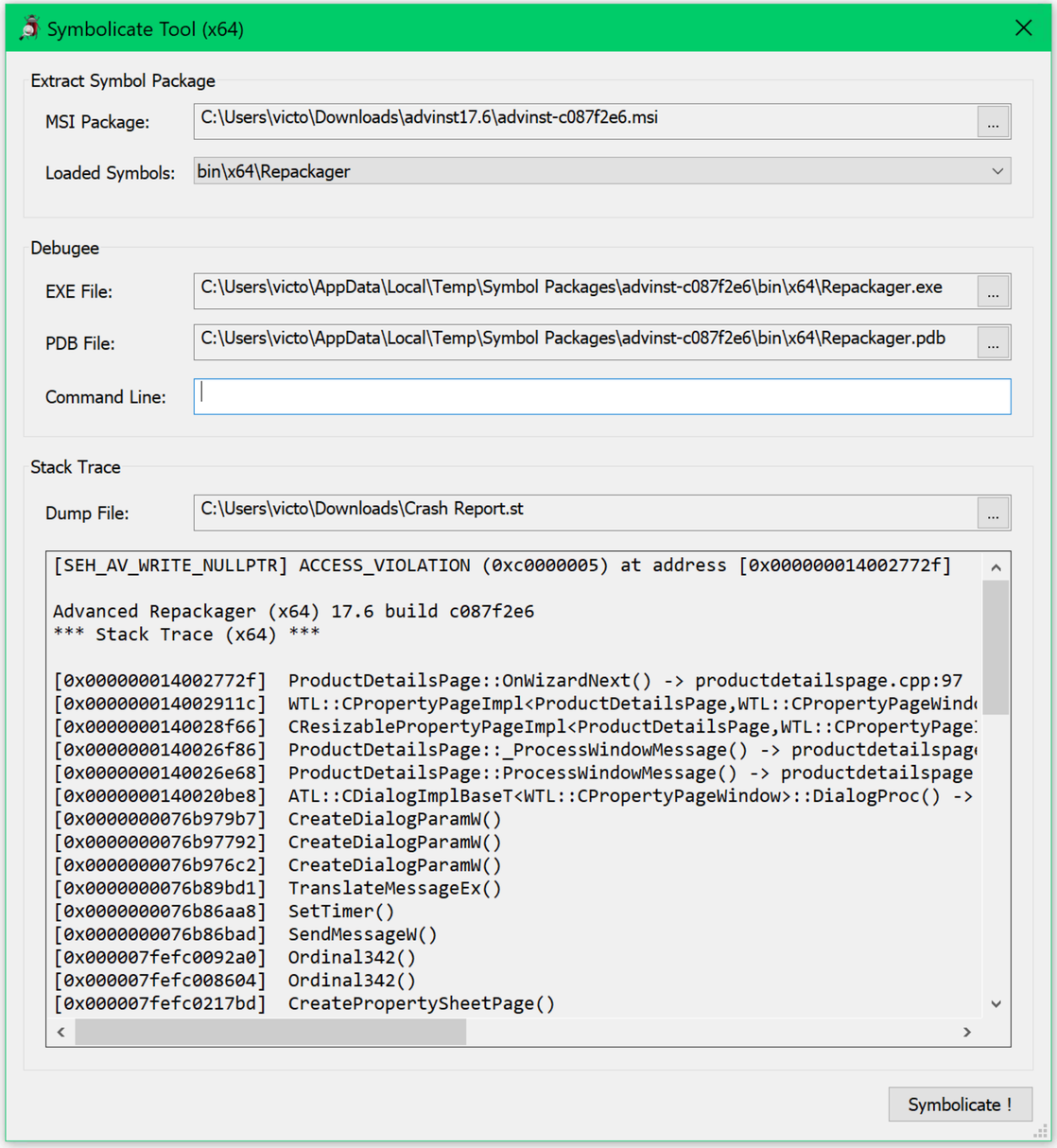


⬅️ select build

Build artifacts

`advinst-c087f2e6.dsym`

Symbolicate Tool



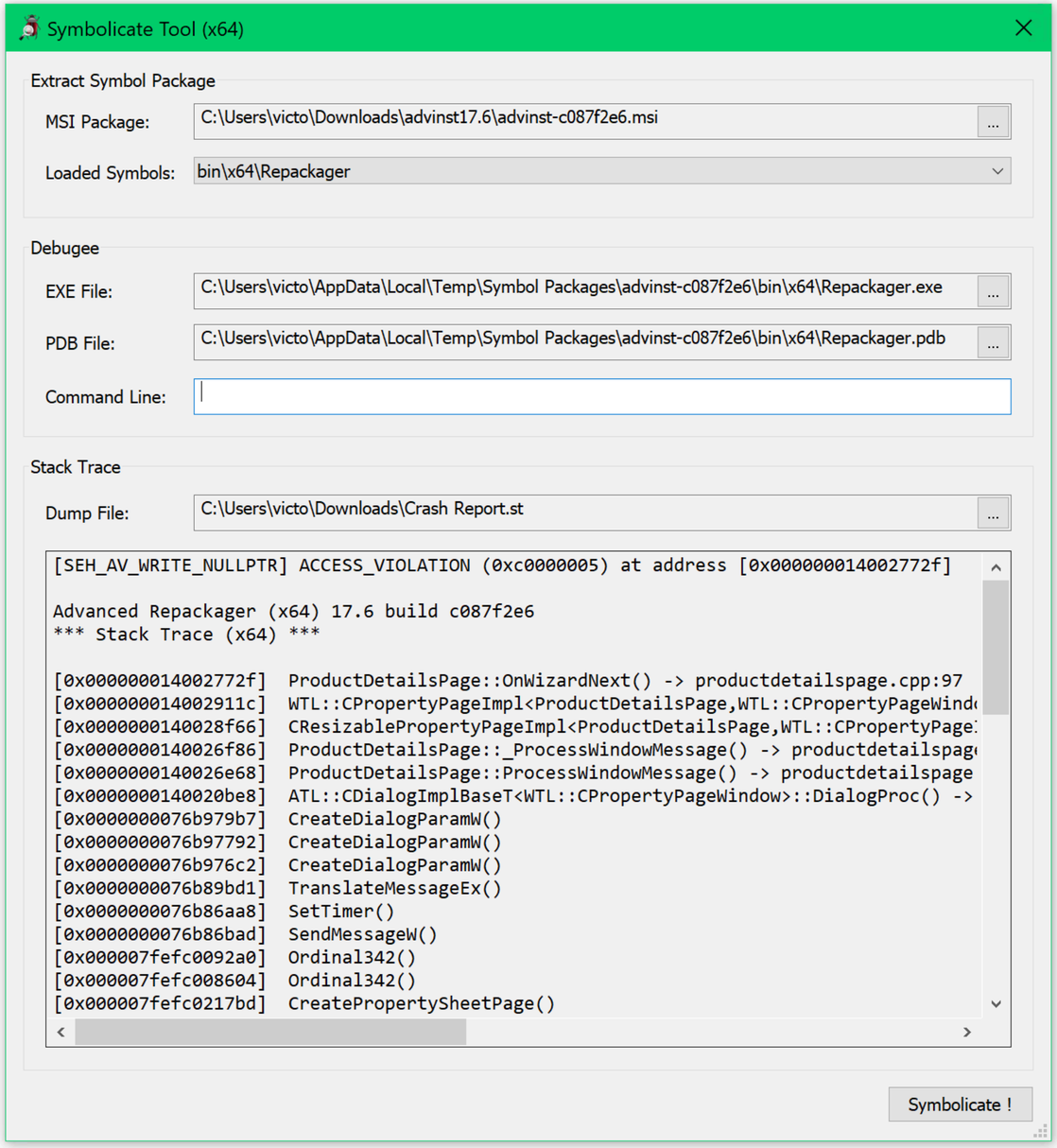
select build

Build artifacts

`advinst-c087f2e6.dsym`

select crash report

Symbolicate Tool



⬅️ select build

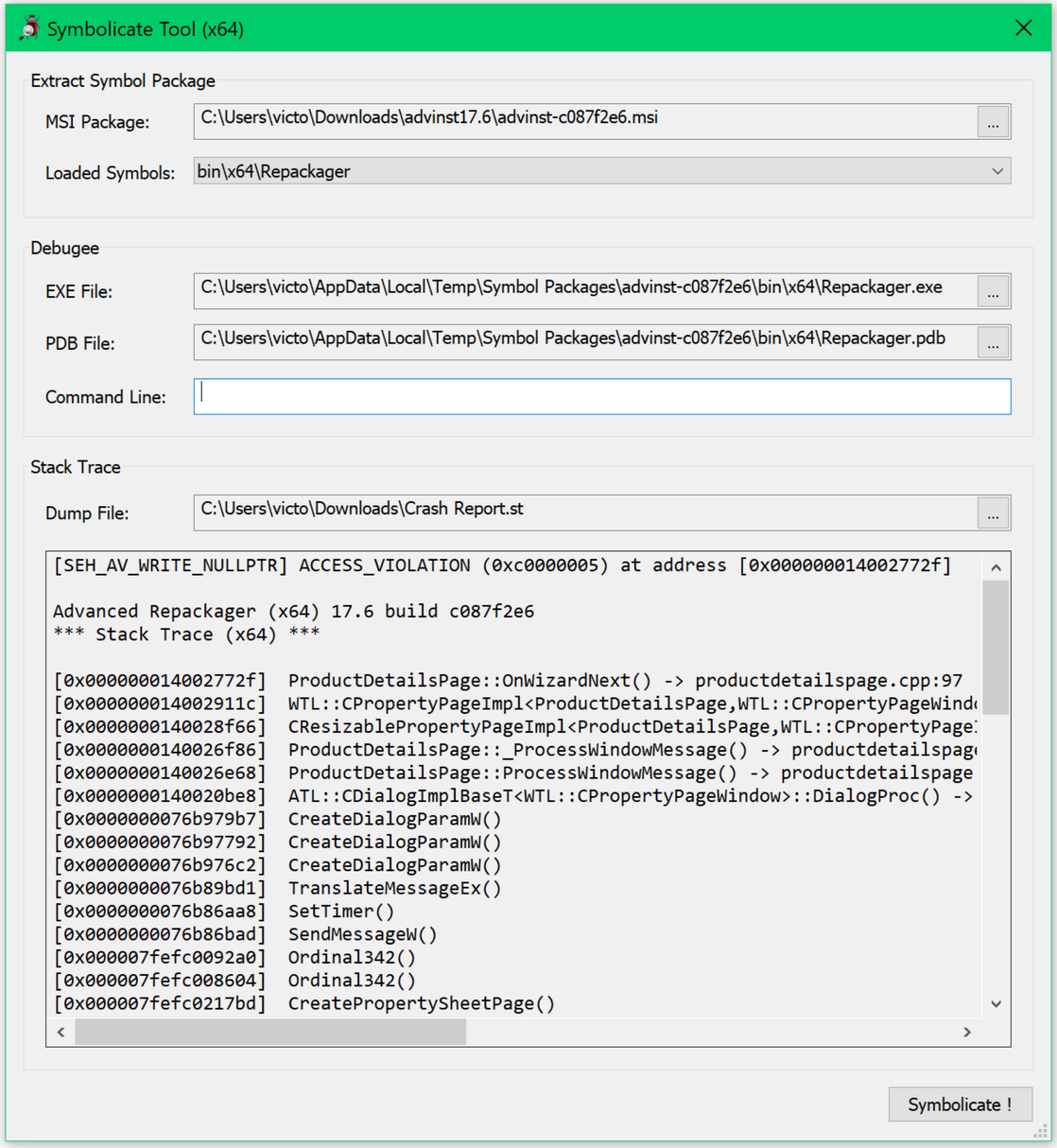
Build artifacts

`advinst-c087f2e6.dsym`

⬅️ select crash report

⬅️ get a full stack trace

Symbolicate Tool



⬅️ select build

Build artifacts

`advinst-c087f2e6.dsym`

⬅️ select crash report

⬅️ get a full stack trace

Fix bug & rejoice!

Symbols

```
[SEH_AV_WRITE_NULLPTR] ACCESS_VIOLATION (0xc0000005) at address [0x000000014002772f]

Advanced Repackager (x64) 17.6 build c087f2e6
*** Stack Trace (x64) ***

[0x000000014002772f]  ProductDetailsPage::OnWizardNext() -> productdetailspage.cpp:97
[0x000000014002911c]  WTL::CPropertyPageImpl<ProductDetailsPage,WTL::CPropertyPageWindow>::OnNotify() -> atldlgs.h:4527
[0x0000000140028f66]  CResizablePropertyPageImpl<ProductDetailsPage>::_ProcessWindowMessage() -> resizablepropsheetimpl.h:443
[0x0000000140026f86]  ProductDetailsPage::_ProcessWindowMessage() -> productdetailspage.h:36
[0x0000000140026e68]  ProductDetailsPage::ProcessWindowMessage() -> productdetailspage.h:31
[0x0000000140020be8]  ATL::CDialogImplBaseT<WTL::CPropertyPageWindow>::DialogProc() -> atlwin.h:3862
[0x0000000076b979b7]  CreateDialogParamW()
[0x0000000076b97792]  CreateDialogParamW()
[0x0000000076b976c2]  CreateDialogParamW()
[0x0000000076b89bd1]  TranslateMessageEx()
[0x0000000076b86aa8]  SetTimer()
[0x0000000076b86bad]  SendMessageW()
[0x000007fefc0092a0]  Ordinal342()
[0x000007fefc008604]  Ordinal342()
[0x000007fefc0217bd]  CreatePropertySheetPage()
[0x000007fefc023075]  CreatePropertySheetPage()
[0x000007fefc023223]  CreatePropertySheetPage()
[0x000007fefc024491]  CreatePropertySheetPage()
[0x0000000076b979b7]  CreateDialogParamW()
[0x0000000076b97792]  CreateDialogParamW()
[0x0000000076b976c2]  CreateDialogParamW()
[0x0000000076b89bd1]  TranslateMessageEx()
[0x0000000076b83bfc]  CallWindowProcW()
[0x0000000076b83b78]  CallWindowProcW()
[0x000000014003268e]  WTL::CPropertySheetImpl<RepackagerWizard,WTL::CWizard97SheetWindow>::OnCommand() -> atldlgs.h:4257
[0x0000000140031f33]  WTL::CWizard97SheetImpl<RepackagerWizard,WTL::CWizard97SheetWindow>::ProcessWindowMessage() -> atldlgs.h:5387
[0x0000000140032257]  CResizablePropSheetImpl<RepackagerWizard>::_ProcessWindowMessage() -> resizablepropsheetimpl.h:138
[0x00000001400312d4]  RepackagerWizard::ProcessWindowMessage() -> repackagerwizard.h:48
[0x00000001400338a3]  ATL::CWindowImplBaseT<WTL::CWizard97SheetWindow,ATL::CWinTraits<1442840576,0> >::WindowProc() -> atlwin.h:3508
[0x0000000076b89bd1]  TranslateMessageEx()
[0x0000000076b86aa8]  SetTimer()
[0x0000000076b86bad]  SendMessageW()
```

Our first clues...

```
[SEH_AV_WRITE_NULLPTR] ACCESS_VIOLATION (0xc0000005) at address [0x000000014002772f]

Advanced Repackager (x64) 17.6 build c087f2e6
*** Stack Trace (x64) ***

[0x000000014002772f]  ProductDetailsPage::OnWizardNext() -> productdetailspage.cpp:97
[0x000000014002911c]  WTL::CPropertyPageImpl<ProductDetailsPage>::OnNotify() -> atldlgs.h:4527
[0x0000000140026f86]  ProductDetailsPage::_ProcessWindowMessage() -> productdetailspage.h:36
[0x0000000140026e68]  ProductDetailsPage::ProcessWindowMessage() -> productdetailspage.h:31
[0x0000000140020be8]  ATL::CDialogImplBaseT<WTL::CPropertyPageWindow>::DialogProc() -> atlwin.h:3862

...

[0x000000014003268e]  WTL::CPropertySheetImpl<RepackagerWizard>::OnCommand() -> atldlgs.h:4257
[0x000000014003312d4]  RepackagerWizard::ProcessWindowMessage() -> repackagerwizard.h:48
[0x000000014003338a3]  ATL::CWindowImplBaseT<WTL::CWizard97SheetWindow>::WindowProc() -> atlwin.h:3508
[0x000000014004176e]  Repackager::RunNormal() -> repackager.cpp:192
[0x000000014003429bb]  wWinMain() -> repackager.cpp:250
[0x0000000140089d02]  __tmainCRTStartup() -> crtexe.c:547
[0x0000000076a6652d]  BaseThreadInitThunk()
[0x000000007715c521]  RtlUserThreadStart()
[0x0000000000a00000]  MODULE_BASE_ADDRESS
```

How it works

Structured Exception Handling (SEH)

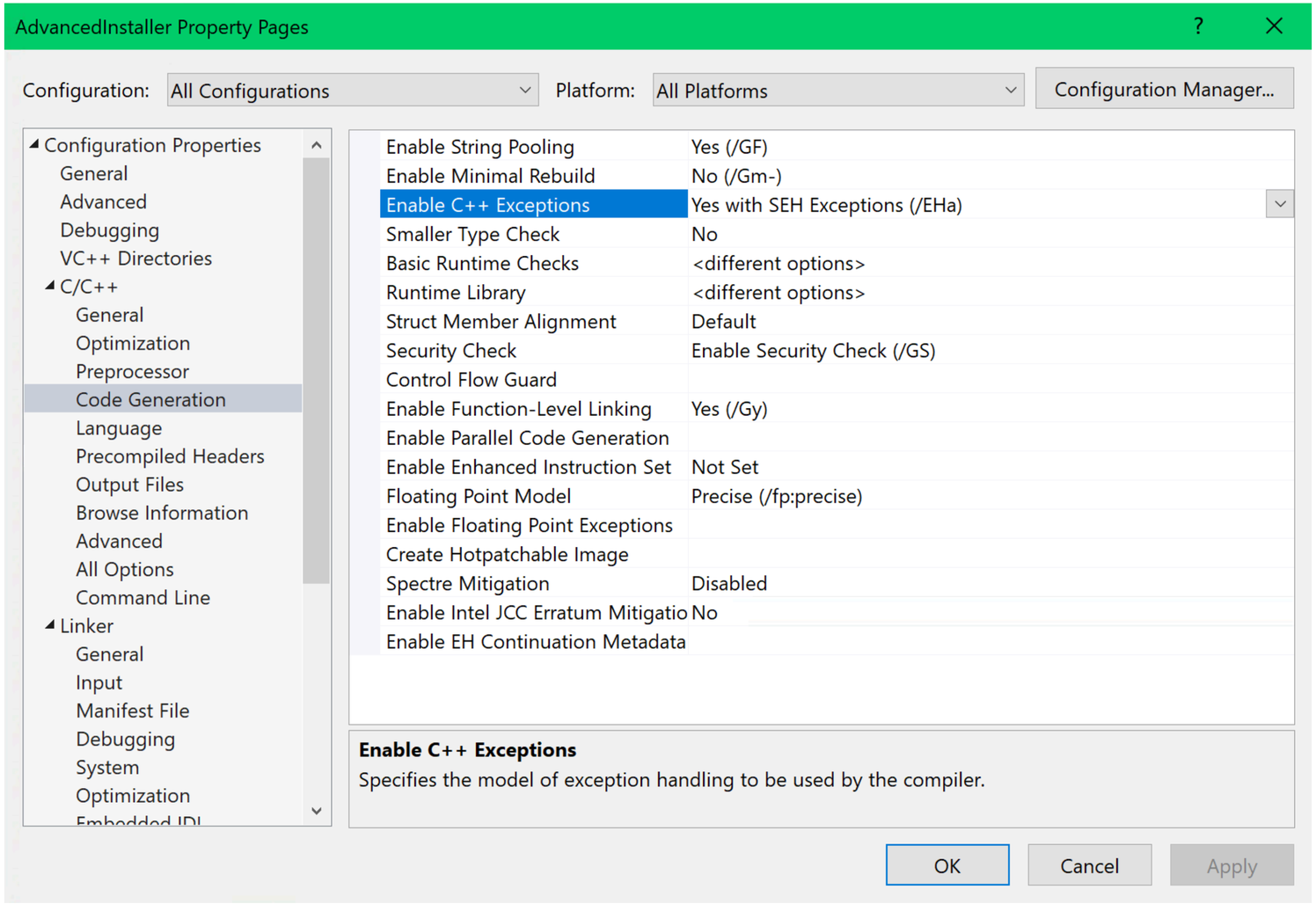
`/EHa`

we use *async* exceptions on all modules

docs.microsoft.com/en-us/windows/win32/debug/structured-exception-handling

docs.microsoft.com/en-us/cpp/cpp/structured-exception-handling-c-cpp?view=msvc-160

Structured Exception Handling (SEH)



Structured Exception Handling (SEH)

```
<ItemDefinitionGroup>
  <ClCompile>
    <DebugInformationFormat>ProgramDatabase</DebugInformationFormat>
    <ExceptionHandling>Async</ExceptionHandling>
  </ClCompile>
  <Link>
    <GenerateDebugInformation>DebugFull</GenerateDebugInformation>
    <SubSystem>Windows</SubSystem>
  </Link>
</ItemDefinitionGroup>
```

/EHa /DEBUG:FULL /Zi

Structured Exception Handling (SEH)

Handle C structured exceptions (Win32) as C++ typed exceptions:

```
_set_se_translator(ExceptionHandling::TransFunc);
```

docs.microsoft.com/en-us/cpp/c-runtime-library/reference/set-se-translator

Unhandled Exceptions

```
static bool installedFilter = false;
if (!installedFilter)
{
  ::SetUnhandledExceptionFilter(ExceptionHandling::UnhandledException);
  installedFilter = true;
}
```

If an exception occurs in a process that is not being debugged,

and the exception makes it to the **Unhandled** exception filter => we intercept it

This replaces the existing top-level exception filter for ALL existing and

ALL future threads in the calling process.

Unhandled Exceptions

```
LONG ExceptionHandling::UnhandledException(EXCEPTION_POINTERS * aExceptionInfo)
{
  wstring message(L"[EXCEPTION_UNHANDLED] ");

  wchar_t buf[MSG_BUFFER_LEN];
  swprintf_s(buf, MSG_BUFFER_LEN, L"(0x%.8x) at address " ADDRESS_FORMAT SW_EOL,
             aExceptionInfo->ExceptionRecord->ExceptionCode,
             aExceptionInfo->ExceptionRecord->ExceptionAddress);
  message += buf;

  StackWalker::TraceFromContext(message, aExceptionInfo->ContextRecord);

  ErrMsgPresenter::Message(message);

  return EXCEPTION_EXECUTE_HANDLER;
}
```

SEH Translator

```
void ExceptionHandling::TransFunc(unsigned int aSECode, EXCEPTION_POINTERS * aExInfo)
{
  // write the exception prolog (type, code, address, etc.)

  switch (aSECode)  // decode SEH exception type
  {
    case EXCEPTION_ACCESS_VIOLATION:
      swprintf_s(buf, MSG_BUFFER_LEN, L"%hs (0x%.8x) at address " ADDRESS_FORMAT SW_EOL,
                 "ACCESS_VIOLATION", EXCEPTION_ACCESS_VIOLATION,
                 aExInfo->ExceptionRecord->ExceptionAddress);
    break;
    case EXCEPTION_DATATYPE_MISALIGNMENT:
    break;
    case EXCEPTION_INT_DIVIDE_BY_ZERO:
    break;
    case EXCEPTION_INT_OVERFLOW:
    break;
    case EXCEPTION_ILLEGAL_INSTRUCTION:
    break;
    case EXCEPTION_STACK_OVERFLOW:
    break;
    ...
```

docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-exception_record

SEH Translator

```
void ExceptionHandling::TransFunc(unsigned int aSECode, EXCEPTION_POINTERS * aExInfo)
{
  ...
  SehException::SEType seType = SehException::SEH_GENERIC;

  // for AV exception, we can determine the type of operation that caused it
  if (aSECode == EXCEPTION_ACCESS_VIOLATION)
  {
    // the first element of the array contains a read-write flag
    // that indicates the type of operation that caused the access violation
    ULONG_PTR operationType = aExInfo->ExceptionRecord->ExceptionInformation[0];

    // the second array element specifies the virtual address of the inaccessible data
    ULONG_PTR virtualAddress = aExInfo->ExceptionRecord->ExceptionInformation[1];

    if (operationType == 0)
      seType = virtualAddress ? SehException::SEH_AV_READ_BADPTR  : SehException::SEH_AV_READ_NULLPTR;
    else if (operationType == 1)
      seType = virtualAddress ? SehException::SEH_AV_WRITE_BADPTR : SehException::SEH_AV_WRITE_NULLPTR;
    else if (operationType == 8)
      seType = virtualAddress ? SehException::SEH_AV_DEP_BADPTR   : SehException::SEH_AV_DEP_NULLPTR;
  }

  // record SEH type info in exception message
  exceptionMsg.insert(0, L"[" + SehException::SeTypeToString(seType) + L"] ");
```

SEH Translator

```
void ExceptionHandling::TransFunc(unsigned int aSECode, EXCEPTION_POINTERS * aExInfo)
{
  // write the exception prolog (type, code, address, etc.)
  // decode SEH exception type
  ...

  // walk the function call stack and gather information about each frame
  StackWalker::TraceFromContext(exceptionMsg, aExInfo->ContextRecord);

  // for AV exception, we can determine the type of operation that caused it
  ... => seType

  // extract SEH exception origin from StackTrace
  SymbolUtil::SrcPos exOrigin = GetExceptionOrigin(aExInfo->ContextRecord);

  // throw a C++ typed exception with the necessary fault information (attached)
  throw SehException(exOrigin.mFile, exOrigin.mLine, seType, exceptionMsg);
}
```

So we end up with a regular C++ exception wrapping the **SEH** info

What's the catch ?

What about an exception **in flight** ?

Get the stack trace for the raised exception on the *current thread*.

What's the catch ?

What about an exception **in flight** ?

Get the stack trace for the raised exception on the *current thread*.

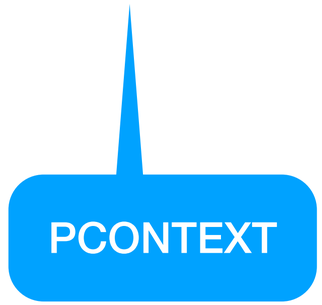
```
wstring ExceptionHandling::GetStackTraceForCurrentException()
{
  wstring stackTrace;
  StackWalker::TraceFromContext(stackTrace, ExceptionHandling::GetCurrentExceptionContext());

  return stackTrace;
}
```

What's the catch ?

What about an exception **in flight** ?

Get the stack trace for the raised exception on the *current thread*.

```
wstring ExceptionHandling::GetStackTraceForCurrentException()
{
  wstring stackTrace;
  StackWalker::TraceFromContext(stackTrace, ExceptionHandling::GetCurrentExceptionContext());

  return stackTrace;
}
```



PCONTEXT

```
typedef struct _CONTEXT {
    DWORD ContextFlags;

    // This section is specified/returned if CONTEXT_DEBUG_REGISTERS is
    // set in ContextFlags.  Note that CONTEXT_DEBUG_REGISTERS is NOT
    // included in CONTEXT_FULL.
    DWORD   Dr0;
    DWORD   Dr1;
    DWORD   Dr2;
    DWORD   Dr3;
    DWORD   Dr6;
    DWORD   Dr7;

    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_FLOATING_POINT.
    FLOATING_SAVE_AREA FloatSave;

    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_SEGMENTS.
    DWORD   SegGs;
    DWORD   SegFs;
    DWORD   SegEs;
    DWORD   SegDs;

    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_INTEGER.
    DWORD   Edi;
    DWORD   Esi;
    DWORD   Ebx;
    DWORD   Edx;
    DWORD   Ecx;
    DWORD   Eax;

    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_CONTROL.
    DWORD   Ebp;
    DWORD   Eip;
    DWORD   SegCs;              // MUST BE SANITIZED
    DWORD   EFlags;             // MUST BE SANITIZED
    DWORD   Esp;
    DWORD   SegSs;

    // This section is specified/returned if the ContextFlags word
    // contains the flag CONTEXT_EXTENDED_REGISTERS.
    // The format and contexts are processor specific
    BYTE    ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];

} CONTEXT;
```

Contains processor-specific **register** data.

The system uses `CONTEXT` structures to perform various internal operations.

docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-context

EXCEPTION_POINTERS >> PCONTEXT

We've already seen this (PCONTEXT)

```
void ExceptionHandling::TransFunc(unsigned int aSECode, EXCEPTION_POINTERS * aExInfo)
{
  StackWalker::TraceFromContext(exceptionMsg, aExInfo->ContextRecord);
  ...
```

```
typedef struct _EXCEPTION_POINTERS
{
  PEXCEPTION_RECORD ExceptionRecord;
  PCONTEXT ContextRecord;
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-exception_pointers

EXCEPTION_RECORD

```
typedef struct _EXCEPTION_RECORD
{
  DWORD ExceptionCode;
  DWORD ExceptionFlags;
  struct _EXCEPTION_RECORD * ExceptionRecord;
  PVOID ExceptionAddress;
  DWORD NumberParameters;
  ULONG_PTR ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD;
```

docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-exception_record

EXCEPTION_RECORD x86/x64

```
typedef struct _EXCEPTION_RECORD32 {
    DWORD    ExceptionCode;
    DWORD ExceptionFlags;
    DWORD ExceptionRecord;
    DWORD ExceptionAddress;
    DWORD NumberParameters;
    DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD32, *PEXCEPTION_RECORD32;


typedef struct _EXCEPTION_RECORD64 {
    DWORD    ExceptionCode;
    DWORD ExceptionFlags;
    DWORD64 ExceptionRecord;
    DWORD64 ExceptionAddress;
    DWORD NumberParameters;
    DWORD __unusedAlignment;
    DWORD64 ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD64, *PEXCEPTION_RECORD64;
```

docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-exception_record

Where to start ?

So, how do we get this `PCONTEXT` ?

#if _MSC_VER >= 1900
(Visual Studio 2015-19)

Where to start ?

So, how do we get this PCONTEXT ?

#if _MSC_VER >= 1900
(Visual Studio 2015-19)

```
PCONTEXT ExceptionHandling::GetCurrentExceptionContext()
{
  __vcrt_ptd * pTid = nullptr;

#ifdef _DLL  // Multi-Threaded DLL /MD or /MDd

  pTid = (__vcrt_ptd *)(((BYTE *)__current_exception_context())
                        - offsetof(__vcrt_ptd, _curcontext));

#else        // Multi-Threaded  /MT or /MTd

  pTid = __vcrt_getptd();

#endif

  return (CONTEXT *)pTid->_curcontext;
}
```

Where to start ?

So, how do we get this `PCONTEXT` ?

#if _MSC_VER < 1900
(Visual Studio 2013)

Where to start ?

So, how do we get this PCONTEXT ?

#if _MSC_VER < 1900
(Visual Studio 2013)

```
PCONTEXT ExceptionHandling::GetCurrentExceptionContext()
{
  _tiddata * pTid = nullptr;

#ifdef _DLL  // Multi-Threaded DLL /MD or /MDd

  pTid = (_tiddata *)(((BYTE *)__pxcptinfoptrs())
                      - offsetof(_tiddata, _tpxcptinfoptrs));

#else        // Multi-Threaded  /MT or /MTd

  pTid = _getptd();

#endif

  return (CONTEXT *)pTid->_curcontext;
}
```

CRT Power

```
#include <eh.h>

#include <signal.h>  // for use of API void ** __pxcptinfoptrs()

#if _MSC_VER >= 1900

  #include <../CRT/src/vcruntime/vcruntime_internal.h>

  extern "C" __vcrt_ptd * __cdecl __vcrt_getptd();
  extern "C" void ** __cdecl __current_exception_context();

#else

  // for use of (private) API _tiddata * _getptd()
  #include <../CRT/src/mtdll.h>

#endif
```

CRT Power

```
// per-thread data
typedef struct __vcrt_ptd  // #include <../CRT/src/vcruntime/vcruntime_internal.h>
{
  // C++ Exception Handling (EH) state
  unsigned long      _NLG_dwCode;       // Required by NLG routines
  unexpected_handler _unexpected;       // unexpected() routine
  void *             _translator;       // S.E. translator
  void *             _purecall;         // called when pure virtual happens
  void *             _curexception;     // current exception
  void *             _curcontext;       // current exception context
  int                _ProcessingThrow;  // for uncaught_exception
  void *             _curexcspec;       // for handling exceptions thrown from std::unexpected
  int                _cxxReThrow;       // true if it's a rethrown C++ exception

#if defined _M_X64 || defined _M_ARM || defined _M_ARM64
  void *    _pExitContext;
  void *    _pUnwindContext;
  void *    _pFrameInfoChain;
  uintptr_t _ImageBase;
  uintptr_t _ThrowImageBase;
  void *    _pForeignException;
#elif defined _M_IX86
  void * _pFrameInfoChain;
#endif
} __vcrt_ptd;
```

Caller PCONTEXT ...

What if we want to get the current StackTrace from the *context of the caller* ?
(on demand - eg. assertions, logging)

When **no exception** is in flight!

How to get the **caller**'s `PCONTEXT` ?

Caller PCONTEXT ...

```
void StackWalker::TraceFromCaller(wstring & aStackMsg)
{
  using PF_RtlCaptureContext = void(WINAPI *)(PCONTEXT aContextRecord);

  // dynamically load the RtlCaptureContext() kernel API
  static auto CaptureCtx = (PF_RtlCaptureContext)::GetProcAddress(
      ::LoadLibraryA("Kernel32.dll"), "RtlCaptureContext");

  CONTEXT context;
  ::ZeroMemory(&context, sizeof(context));

  // retrieve the context record of the caller function
  CaptureCtx(&context);

  StackWalker::TraceFromContext(aStackMsg, &context);
}
```

PCONTEXT ...

So now we know how to get this PCONTEXT

How do we *walk the stack* ?

```
wstring stackTrace;
StackWalker::TraceFromContext(stackTrace, GetCurrentExceptionContext());
```

Walk the stack - init

```
void StackWalker::TraceFromContext(wstring & aStackMsg, PCONTEXT aContext, int MaxFrameCount)
{
  // All <DbgHelp> functions, such as StackWalk(), are single threaded.
  // (calls from more than one thread to this function will likely result
  //  in unexpected behavior or memory corruption)
  // => we must synchronize all concurrent calls to this function
  SyncGuard guard(sEHSyncSupport);

  // Copy the given machine CONTEXT structure because the StackWalk() API
  // might modify it and subsequent calls needing the CONTEXT will fail
  CONTEXT context;
  ::CopyMemory(&context, aContext, sizeof(context));

  HANDLE hProcess = ::GetCurrentProcess();
  HANDLE hThread  = ::GetCurrentThread();

  // create a symbol explorer
  SymbolUtil symMgr;
  if (!symMgr.Init(hProcess))
    return;

  ...
```

Walk the stack

```
void StackWalker::TraceFromContext(wstring & aStackMsg, PCONTEXT aContext, int MaxFrameCount)
{
  ...

  // initialize the STACKFRAME according to the platform we are working on (PE type)
  STACKFRAME sf;
  DWORD imageType = InitStackFrameFromContext(&sf, &context);

  for (int frmIndex = 0; frmIndex < MaxFrameCount; frmIndex++)
  {
    // get the current frame info
    BOOL result = ::StackWalk(imageType, hProcess, hThread, &sf, &context, nullptr,
                              SymFunctionTableAccess, SymGetModuleBase, nullptr);
    if (!result)
      break;

    aStackMsg += symMgr.ComposeStackFrame(sf.AddrPC.Offset);
  }

  // write the module load address - needed because of ASLR (Address Space Layout Randomization)
  aStackMsg += symMgr.ComposeModuleBaseAddress();
}
```

Walk the stack - frame setup

```
DWORD InitStackFrameFromContext(LPSTACKFRAME aStackFrame, PCONTEXT aContext)
{
  ::ZeroMemory(aStackFrame, sizeof(STACKFRAME));

#if defined _M_IX86

  DWORD imageType = IMAGE_FILE_MACHINE_I386;

  aStackFrame->AddrStack.Offset = aContext->Esp;
  aStackFrame->AddrStack.Mode   = AddrModeFlat;

  aStackFrame->AddrFrame.Offset = aContext->Ebp;
  aStackFrame->AddrFrame.Mode   = AddrModeFlat;

  aStackFrame->AddrPC.Offset    = aContext->Eip;
  aStackFrame->AddrPC.Mode      = AddrModeFlat;

#elif defined _M_X64

  .....

#endif

  return imageType;
}
```

Walk the stack - frame setup

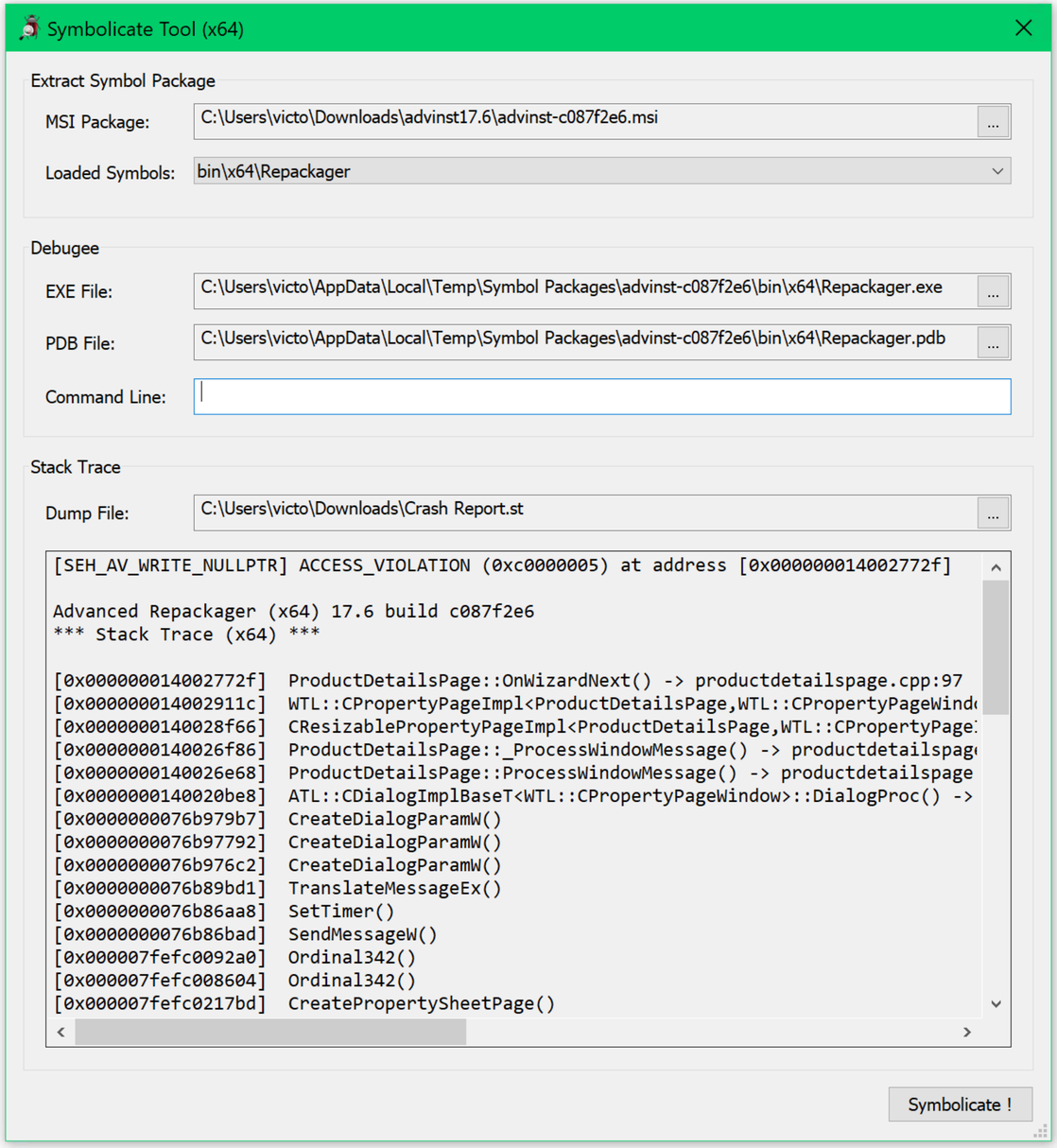
```
DWORD InitStackFrameFromContext(LPSTACKFRAME aStackFrame, PCONTEXT aContext)
{
  ::ZeroMemory(aStackFrame, sizeof(STACKFRAME));

#if defined _M_IX86

  DWORD imageType = IMAGE_FILE_MACHINE_I386;

  aStackFrame->AddrStack.Offset = aContext->Esp;
  aStackFrame->AddrStack.Mode   = AddrModeFlat;

  aStackFrame->AddrFrame.Offset = aContext->Ebp;
  aStackFrame->AddrFrame.Mode   = AddrModeFlat;

  aStackFrame->AddrPC.Offset    = aContext->Eip;
  aStackFrame->AddrPC.Mode      = AddrModeFlat;

#elif defined _M_X64

  .....

#endif

  return imageType;
}
```

```
#elif defined _M_X64

  DWORD imageType = IMAGE_FILE_MACHINE_AMD64;

  aStackFrame->AddrStack.Offset = aContext->Rsp;
  aStackFrame->AddrStack.Mode   = AddrModeFlat;

  aStackFrame->AddrFrame.Offset = aContext->Rbp;
  aStackFrame->AddrFrame.Mode   = AddrModeFlat;

  aStackFrame->AddrPC.Offset    = aContext->Rip;
  aStackFrame->AddrPC.Mode      = AddrModeFlat;

#endif
```

ASLR (Address Space Layout Randomization)

```
// Serialize module base address - needed because of ASLR

wstring SymbolUtil::ComposeModuleBaseAddress()
{
  wstring stackFrame;

  HMODULE moduleLoadAddress = ::GetModuleHandle(nullptr);

  wchar_t buf[MAX_PATH];
  swprintf_s(buf, MAX_PATH, ADDRESS_FORMAT L"  ", (size_t) moduleLoadAddress);

  stackFrame += buf;
  stackFrame += SW_MODULE_LOAD_ADDRESS;
  stackFrame += SW_EOL;
  return stackFrame;
}
```

Symbolicate Tool



⬅️ select build

Build artifacts

`advinst-c087f2e6.dsym`

⬅️ select crash report

⬅️ get a full stack trace

Symbolicate Tool: arch PE/COFF

Symbolicate tool comes in two ISAs: `x86 & x64`

👉 must match the arch of the debugged process

```
CoffBrowse imageInfo(targetExePath);

if (!imageInfo.IsValidPE())
{
  Error("The selected file is not a valid Windows application.");
  return 0;
}

StackTraceAnalyzer analyzer(imageInfo.Is64());

#if defined _M_IX86
  if (imageInfo.Is64())
#elif defined _M_X64
  if (!imageInfo.Is64())
#endif
  {
    Error("Process architecture mismatch (x86/x64). Launch the appropriate version of this tool.");
    return 0;
  }
```

Symbolicate Tool: launch debug session

```
// launch the process we want to debug (SEE_MASK_FLAG_NO_UI)
mDebugProcess = DebugProcessLaunch(exePath, exeCmd);

...

// get the load address of the debugged process - needed because of ASLR
// (Address Space Layout Randomization)
mDebugProcessBaseAddress = (DWORD_PTR)GetMainModuleForProcess(mDebugProcess);

// initialize the symbol handler for the process
mSymMgr.Init(mDebugProcess, symbolsPath);

...

// compose the symbolicated stack trace from the client crash report
mStackTrace = analyzer.Symbolicate(crashReport);
```

::EnumProcessModules()

Symbol INIT

```
bool SymbolUtil::Init(HANDLE aProcess, const wstring & aSymbolsPath)
{
  // set debugging tools global options
  ::SymSetOptions(SYMOPT_UNDNAME | SYMOPT_DEFERRED_LOADS |
                  SYMOPT_LOAD_LINES | SYMOPT_DEBUG);

  // check if we can use debugging tools
  if (!IsAvailable())
    return false;

  // initialize the symbol handler for the given process
  BOOL symInit = ::SymInitialize(mProcess, nullptr, TRUE);
  assert(symInit == TRUE);
  if (!symInit)
    return false;

  // set the process module folder into the search paths for image symbols
  return SetSymbolSearchPath(aSymbolsPath);
}
```

docs.microsoft.com/en-us/windows/win32/api/dbghelp/nf-dbghelp-symsetoptions

Symbol INIT

```
bool SymbolUtil::Init(HANDLE aProcess, const wstring & aSymbolsPath)
{
  // set debugging tools global options
  ::SymSetOptions(SYMOPT_UNDNAME | SYMOPT_DEFERRED_LOADS |
                  SYMOPT_LOAD_LINES | SYMOPT_DEBUG);

  // check if we can use debugging
  if (!IsAvailable())
    return false;

  // initialize the symbol handler for the given process
  BOOL symInit = ::SymInitialize(mProcess, nullptr, TRUE);
  assert(symInit == TRUE);
  if (!symInit)
    return false;

  // set the process module folder into the search paths for image symbols
  return SetSymbolSearchPath(aSymbolsPath);
}
```

```
PF_SymFromAddr DynSymFromAddr() {
  static auto symProc = (PF_SymFromAddr)::GetProcAddress(
    ::LoadLibraryA("Dbghelp.dll"), "SymFromAddr");
  return symProc;
}
```

docs.microsoft.com/en-us/windows/win32/api/dbghelp/nf-dbghelp-symsetoptions

Symbol Search Path

```
bool SymbolUtil::SetSymbolSearchPath(wstring symbolSearchPath)
{
  if (symbolSearchPath.empty())
  {
    // set the process module folder into the search paths for image symbols
    char modulePath[MAX_PATH];
    ::GetModuleFileName(nullptr, modulePath, MAX_PATH);

    symbolSearchPath = modulePath;
    const size_t pos = symbolSearchPath.find_last_of('\\');
    if (pos != string::npos)
      symbolSearchPath = symbolSearchPath.substr(0, pos);
  }

  // set the search paths for symbols (PDB)
  return ::SymSetSearchPath(mProcess, symbolSearchPath.c_str()) == TRUE;
}
```

Symbolicate Tool

```
wstring StackTraceAnalyzer::Symbolicate(const wstring & aRawCrashReport)
{
  ...

  auto stackFrames = GetFrames(aRawCrashReport);

  // extract stack trace BaseAddress (should be the last line in the trace)
  mStackTraceBaseAddress = GetFrameAddress(stackFrames[0]);

  ...

  // symbolicate each frame
  for (const auto & frame : stackFrames)
    processedStackTrace += ProcessStackFrame(frame);

  return processedStackTrace;
}
```

Symbolicate Tool: Process Stack Frame

```
wstring StackTraceAnalyzer::ProcessStackFrame(const StackFrame & aStackFrame)
{
  ...
  // extract stack frame address to symbolicate
  DWORD_PTR frameAddress = GetFrameAddress(aStackFrame);

  if (aStackFrame == SW_MODULE_LOAD_ADDRESS)
    return; // don't translate module base address
  if (aStackFrame != SW_NO_SYMBOL)
    return; // only symbolicate frames with no symbols

  // REBASE: perform address translation
  // needed because of ASLR (Address Space Layout Randomization)
  DWORD_PTR realAddress = frameAddress +
                          (mDebugProcessBaseAddress - mStackTraceBaseAddress);

  // symbolicate stack frame with the debug information
  // of the instrumented process
  return mSymMgr.ComposeStackFrame(realAddress);
}
```

Walk the stack - SymbolUtil

```
wstring SymbolUtil::ComposeStackFrame(DWORD_PTR aAddress)
{
  wstring stackFrame;

  wchar_t buf[MAX_PATH];
  swprintf_s(buf, MAX_PATH, ADDRESS_FORMAT L"  ", aAddress);
  stackFrame += buf;

  // retrieve symbol name
  stackFrame += SymbolNameFromAddress(aAddress);

  // source filename:line
  const wstring & origin = SymbolSourceFromAddress(aAddress);
  if (!origin.empty())
    stackFrame += origin;

  return stackFrame;
}
```

Walk the stack - SymbolUtil

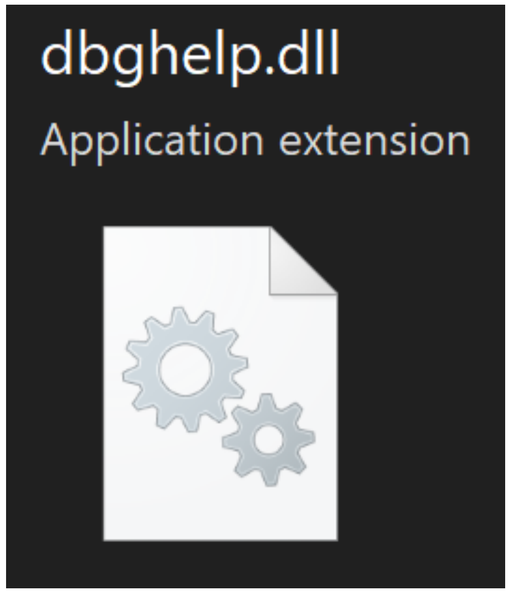
```
wstring SymbolUtil::SymbolNameFromAddress(DWORD_PTR aAddress) const
{
  ...

  auto pSymbol = reinterpret_cast<PSYMBOL_INFO>(mSymMemBuffer);
  pSymbol->SizeOfStruct = sizeof(SYMBOL_INFO);

  // get symbol name (de-mangled function name)
  if (DynSymFromAddr(mProcess, aAddress, nullptr, pSymbol))
    return pSymbol->Name;
  else
    return SW_NO_SYMBOL;
}
```

```
PF_SymFromAddr DynSymFromAddr()
{
  static auto symProc = (PF_SymFromAddr)::GetProcAddress(
    ::LoadLibraryA("Dbghelp.dll"), "SymFromAddr");

  return symProc;
}
```

Walk the stack - SymbolUtil

```
wstring SymbolUtil::SymbolSourceFromAddress(DWORD_PTR aAddress) const
{
  DWORD displacement = 0;

  IMAGEHLP_LINE line;
  ::ZeroMemory(&line, sizeof(line));
  line.SizeOfStruct = sizeof(line);

  // get location information for symbol "sourceFile:lineNo"
  if (::SymGetLineFromAddr(mProcess, aAddress, &displacement, &line))
  {
    wchar_t origin[MAX_PATH];
    swprintf_s(origin, MAX_PATH, L"%s:%ld", line.FileName, line.LineNumber);
    return origin;
  }
}
```

<dbghelp.h>



x86/x64

```
#include <dbghelp.h>
```

/LINK `Dbghelp.lib`

Dynamic dependency on `Dbghelp.dll`

An good in-the-box alternative

VS Snapshot File

Game changer!

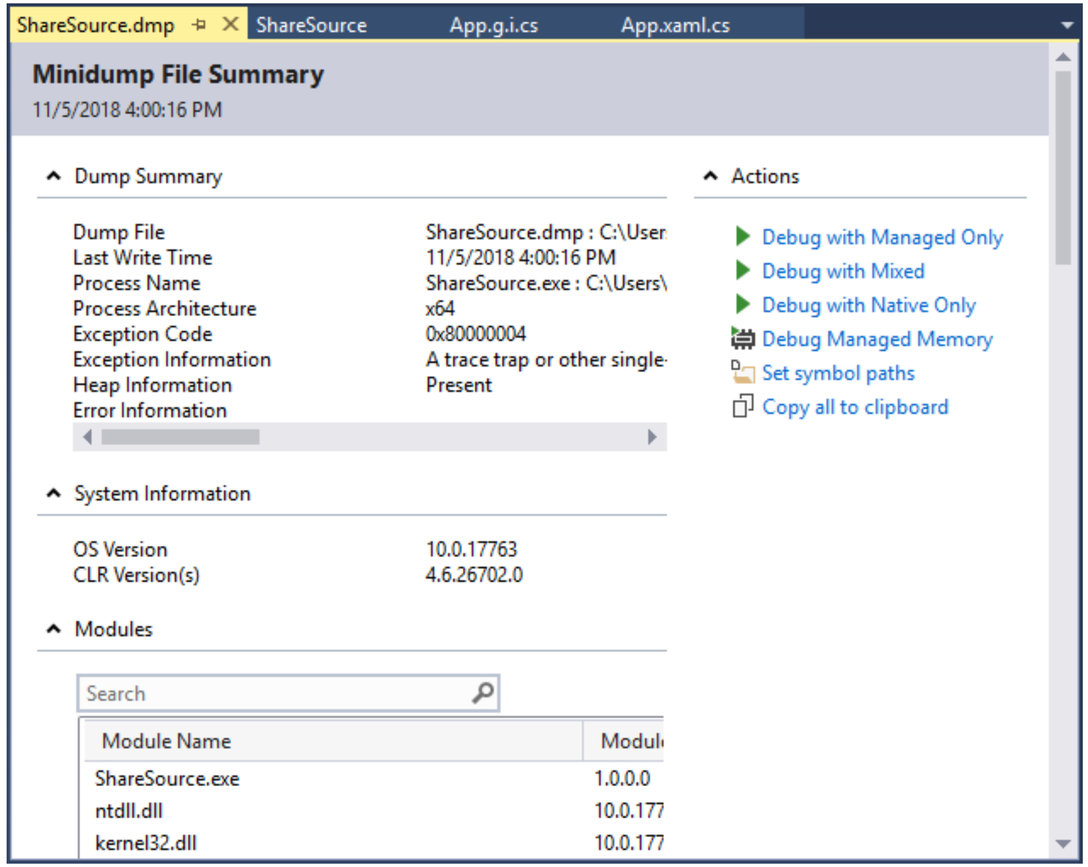
`Minidump file (*.dmp)` <= Windows snapshot process
(program virtual memory/heap + metadata)

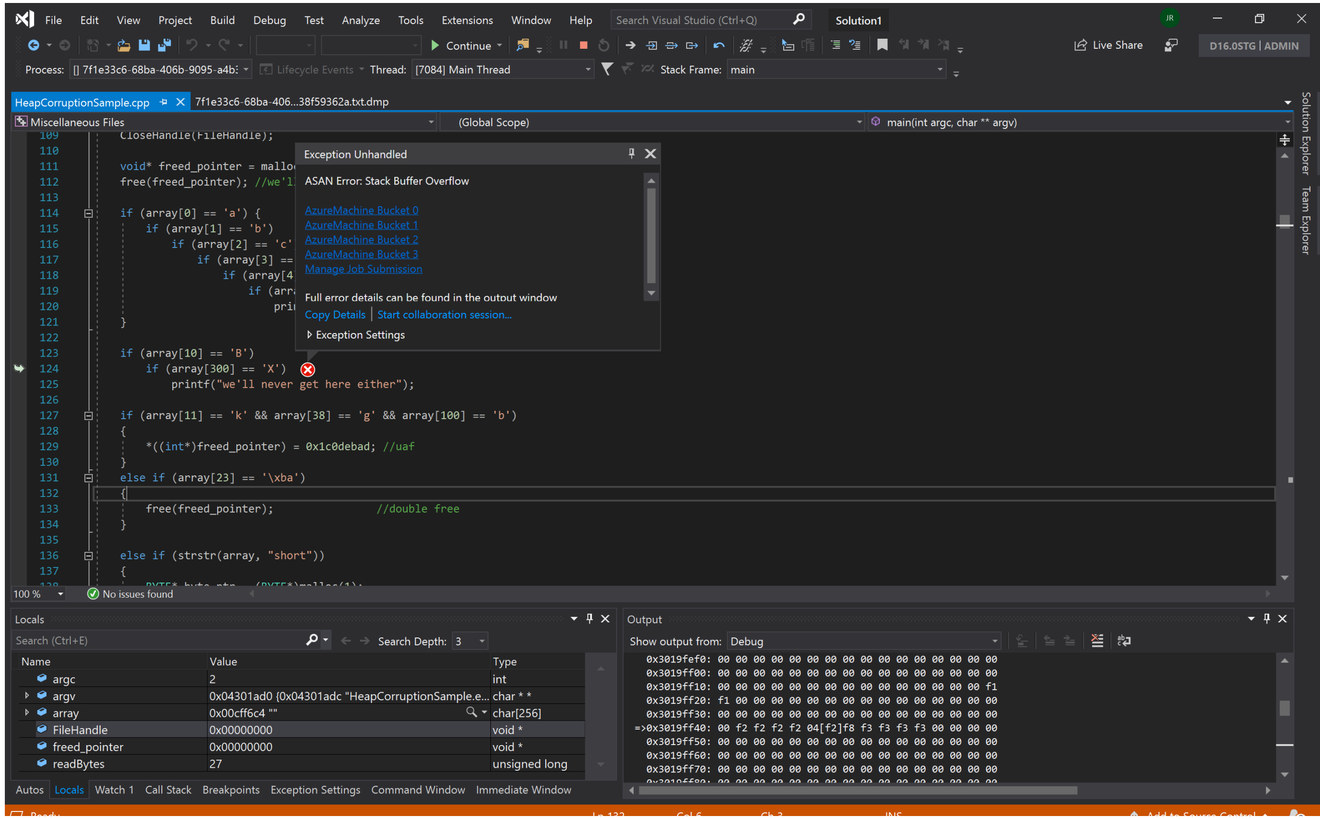
VS can parse & open this => Points at the location the error occurred.

Changes the way you report a bug, in general

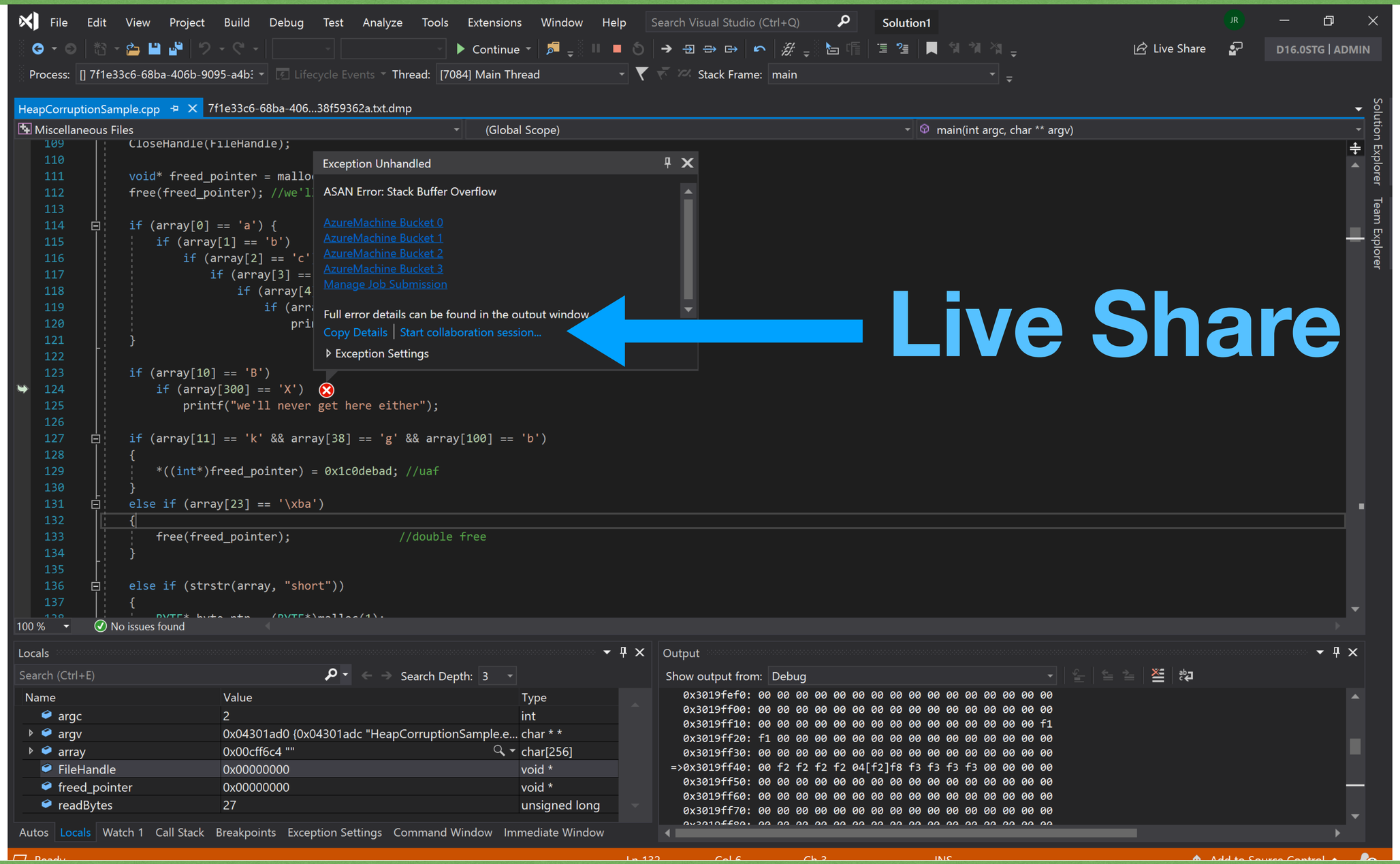
+ Live Share

VS Snapshot File





Snapshot Loaded



Part III

Post-pandemic crashes

`P0881`

A Proposal to add stacktrace library

- Alexey Gorgurov, Antony Polukhin

First draft: 2018-01-23 `[R0]` based on `Boost.Stacktrace`

....

Didn't make into **C++20** 😕

...

`[R7]` wg21.link/P0881

```
#include <stacktrace>
```

Naming is hard

- 10 stacktrace
- 8 backtrace
- 1 back_trace
- 1 stack_trace
- 5 traceback
- 3 call_stack
- 4 call_trace
- 1 call_history

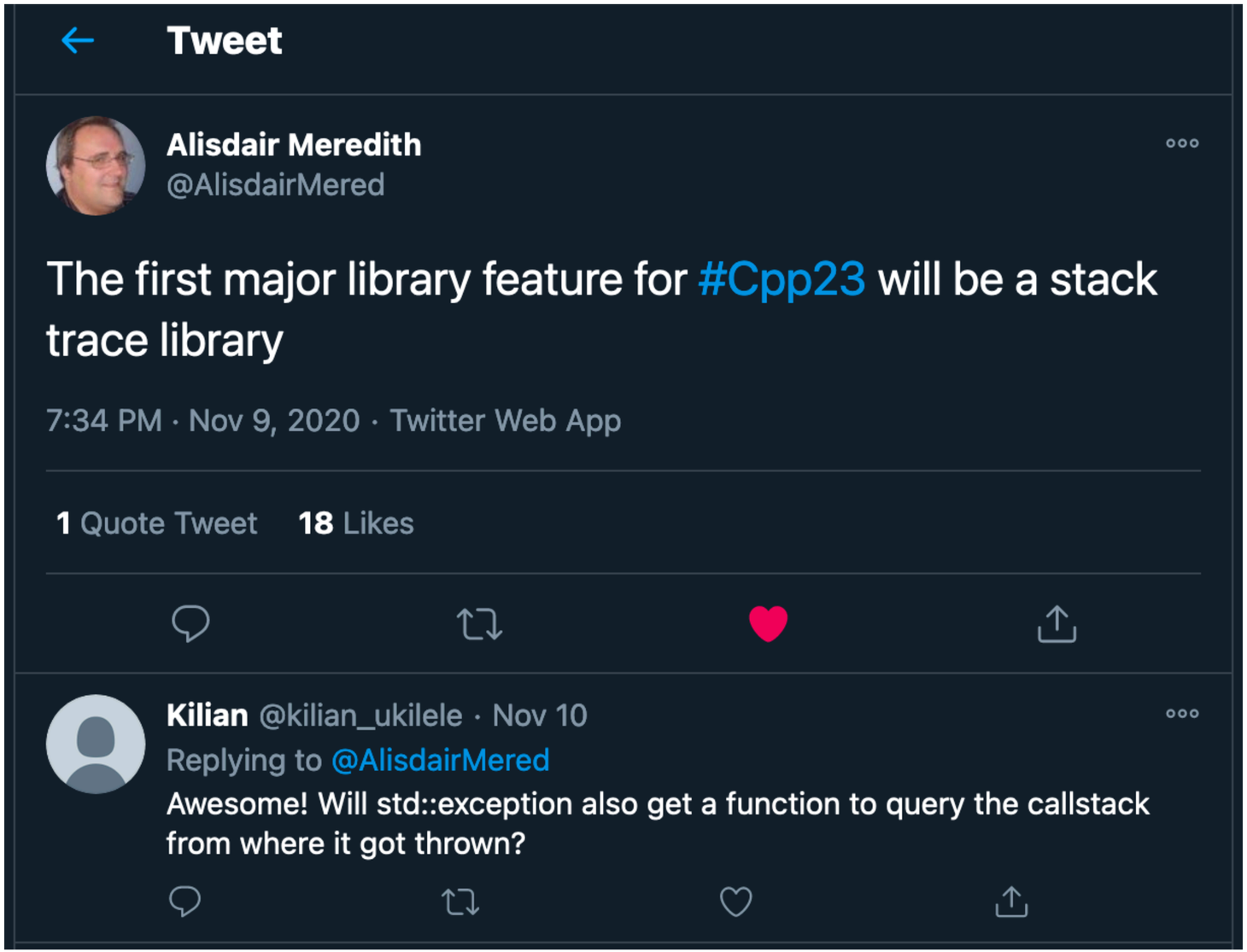
github.com/cplusplus/papers/issues/119

Naming is hard

- 2 stack_frame
- 3 invocation_info
- 9 stacktrace::entry
- 3 stacktrace::frame
- 5 stacktrace_entry
- 1 stacktrace_frame
- 2 frame_descriptor
- 4 call_info
- 2 call_descriptor
- 2 frame_info

github.com/cplusplus/papers/issues/119

C++ 23



👉

twitter.com/alisdairmered/status/1325854252338716672?s=21

C++ 23 <stacktrace>

Key features (desired):

C++ 23 <stacktrace>

Key features (desired):

C++ 23 <stacktrace>

Key features (desired):

- all functions are **lazy**: do not query the stacktrace entry info without explicit request

C++ 23 <stacktrace>

Key features (desired):

- all functions are **lazy**: do not query the stacktrace entry info without explicit request
- **dynamic size** for trace - all the available invokers must be stored in a stacktrace

C++ 23 <stacktrace>

Key features (desired):

- all functions are **lazy**: do not query the stacktrace entry info without explicit request
- **dynamic size** for trace - all the available invokers must be stored in a stacktrace
- implementations: allow to **disable/enable** gathering stacktraces by a linker switch

C++ 23 <stacktrace>

Key features (desired):

- all functions are **lazy**: do not query the stacktrace entry info without explicit request
- **dynamic size** for trace - all the available invokers must be stored in a stacktrace
- implementations: allow to **disable/enable** gathering stacktraces by a linker switch
- stacktracing shouldn't prevent any of **optimizations**

C++ 23 <stacktrace>

Key features (desired):

- all functions are **lazy**: do not query the stacktrace entry info without explicit request
- **dynamic size** for trace - all the available invokers must be stored in a stacktrace
- implementations: allow to **disable/enable** gathering stacktraces by a linker switch
- stacktracing shouldn't prevent any of **optimizations**
- stacktrace should be **usable** in contract violation handler, coroutines, handler functions, parallel algorithms

C++ 23 <stacktrace>

Key features (desired):

C++ 23 <stacktrace>

Key features (desired):

C++ 23 <stacktrace>

Key features (desired):

- `stacktrace_entry::description()` should return a **demangled** function signature

C++ 23 <stacktrace>

Key features (desired):

- `stacktrace_entry::description()` should return a **demangled** function signature
- `to_string(stacktrace)` should query information from **debug symbols**, symbol export tables and any other sources, returning *demangled* signatures

C++ 23 <stacktrace>

Key features (desired):

- `stacktrace_entry::description()` should return a **demangled** function signature
- `to_string(stacktrace)` should query information from **debug symbols**, symbol export tables and any other sources, returning *demangled* signatures
- information about **inlined functions** that have no separate stacktrace entries is welcomed -> `to_string(stacktrace)`

C++ 23 <stacktrace>

Key features (desired):

- `stacktrace_entry::description()` should return a **demangled** function signature
- `to_string(stacktrace)` should query information from **debug symbols**, symbol export tables and any other sources, returning *demangled* signatures
- information about **inlined functions** that have no separate stacktrace entries is welcomed -> `to_string(stacktrace)`
- avoid doing **heavy** operations in `basic_stacktrace` constructors or `stacktrace_entry::current()`

C++ 23 <stacktrace>

```
class stacktrace_entry
{
  public:
    using native_handle_type = implementation-defined;
    ...
    constexpr native_handle_type native_handle() const noexcept;
    constexpr explicit operator bool() const noexcept;
    ...
    string   description() const;
    string   source_file() const;
    uint32_t source_line() const;
};
```

`stacktrace_entry` models concepts:
regular and `three_way_comparable<strong_ordering>`

C++ 23 <stacktrace>

```
template<class Allocator>
class basic_stacktrace
{
public:
  using value_type = stacktrace_entry;
  using allocator_type = Allocator;
  ...
  const_iterator begin() const noexcept;
  const_iterator end() const noexcept;
  const_reverse_iterator rbegin() const noexcept;
  const_reverse_iterator rend() const noexcept;
  ...

private:
  vector<value_type, allocator_type> m_frames;
};
```

C++ 23 <stacktrace>

```
static basic_stacktrace current(const allocator_type& alloc = allocator_type()) noexcept;

static basic_stacktrace current(size_type skip,
                                const allocator_type& alloc = allocator_type()) noexcept;

static basic_stacktrace current(size_type skip, size_type max_depth,
                                const allocator_type& alloc = allocator_type()) noexcept;
```

=> `basic_stacktrace` object with `m_frames` storing the stack trace of the current evaluation in the *current thread* of execution

`alloc` is passed to the constructor of the `m_frames` object.

C++ 23 <stacktrace>

```
namespace std {

  using stacktrace = basic_stacktrace<allocator<stacktrace_entry>>;

  string to_string(const stacktrace_entry& f);

  template<class Alloc>
  string to_string(const basic_stacktrace<Alloc>& st);

  template<class charT, class traits>
  basic_ostream<charT, traits>&
  operator<<(basic_ostream<charT, traits>& os, const stacktrace_entry& f);

  template<class charT, class traits, class Alloc>
  basic_ostream<charT, traits>&
  operator<<(basic_ostream<charT, traits>& os, const basic_stacktrace<Alloc>& st);
```

```
description()
source_file()
source_line()
```

C++ 23 Example

```
auto trace = basic_stacktrace::current();

for (stacktrace_entry frame : trace)
{
  std::cerr << frame.description() << " at "
            << frame.source_file() << ":" << frame.source_line() << "\n";
}
```

C++ 23 Example

```
auto trace = basic_stacktrace::current();

for (stacktrace_entry frame : trace)
{
  std::cerr << std::to_string(frame) << "\n";
}
```

C++ 23 Example

```
auto trace = basic_stacktrace::current();

for (stacktrace_entry frame : trace)
{
  std::cerr << frame << "\n";
}
```

C++ 23 Example

```
auto trace = basic_stacktrace::current();

std::cerr << std::to_string(trace);
```

C++ 23 Example

```
auto trace = basic_stacktrace::current();

std::cerr << trace;
```

C++ 23 <stacktrace>

It can't get any simpler than that.

C++ 23 <stacktrace>

It can't get any simpler than that.

I can't wait to see **early** implementations from our standard library providers!

Q & A

The Quest For A Better Crash

C++ Now 2021

May 7


@ciura_victor

Victor Ciura
Principal Engineer


CAPHYON