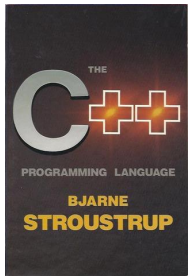
Windows Debug Info: CodeView and PDB Files

Hans Wennborg
StockholmCpp
22 October 2026

Symbolic Debug Information



Compiler

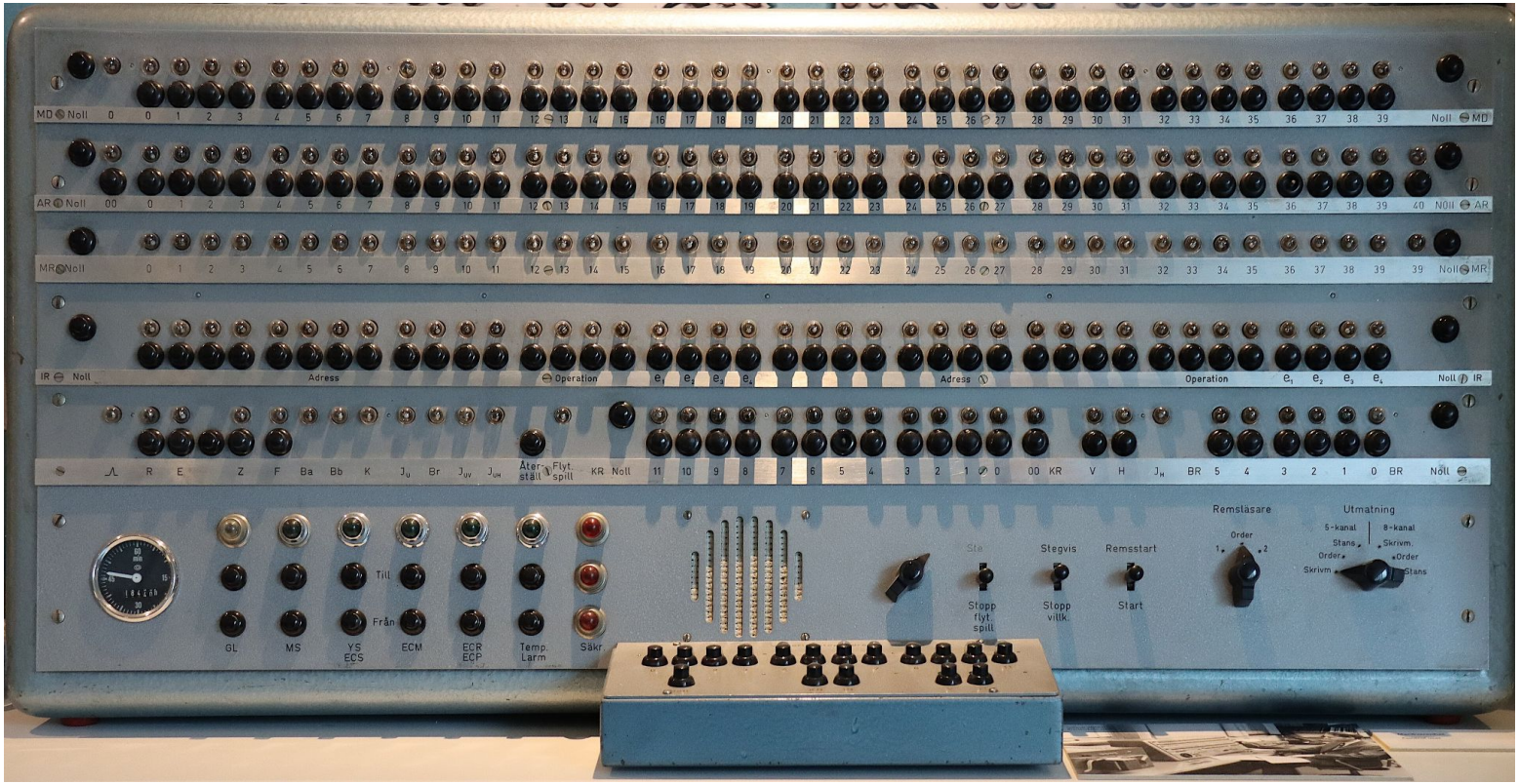
Machine code



Debug info

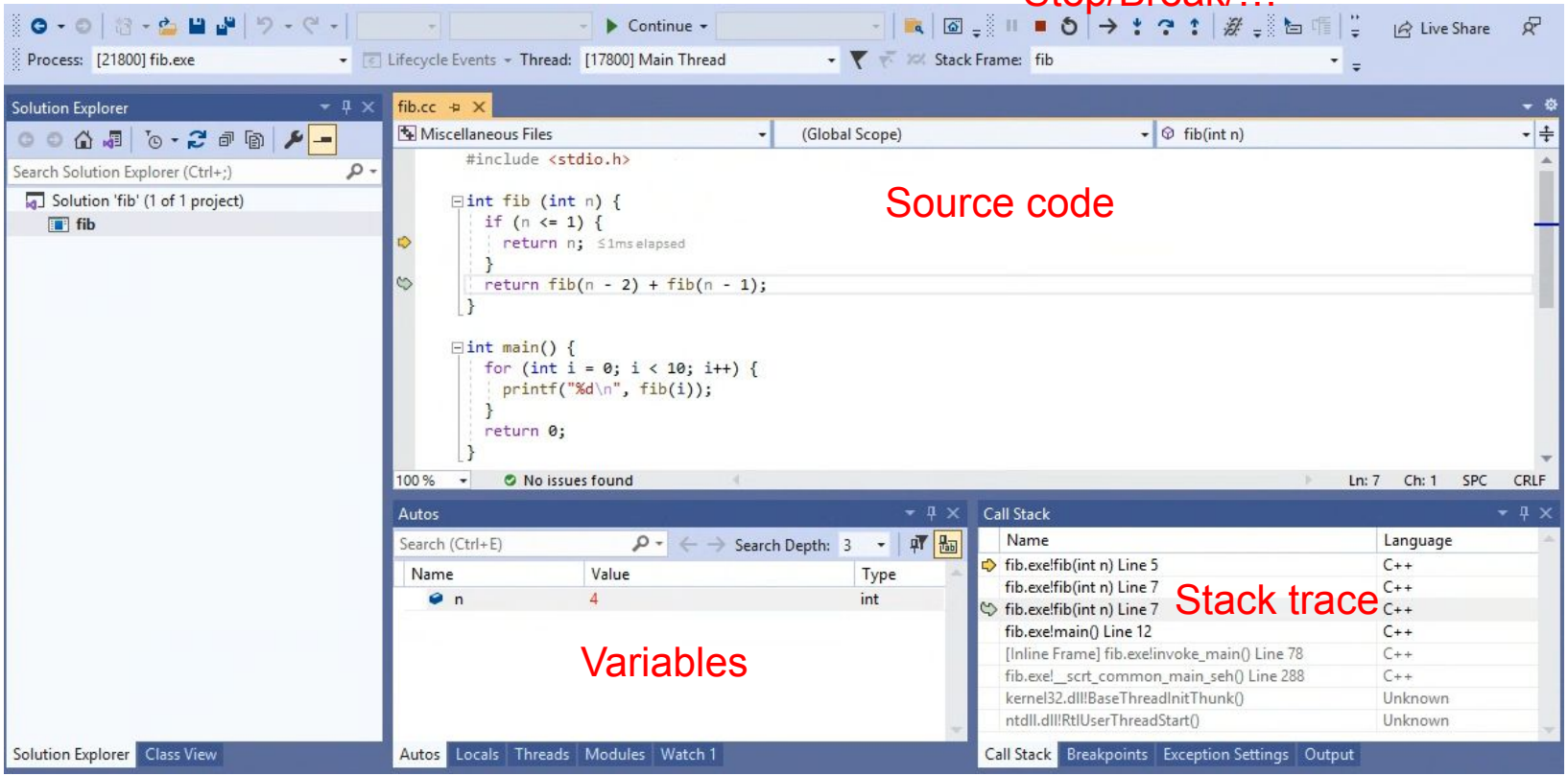
Debugger

Machine-Level Debugging



https://en.wikipedia.org/wiki/SMIL_(computer)
Nixdorf, CC BY-SA 4.0

Source-Level / Symbolic Debugging



Symbolic Debug Information

- Symbol table, and possibly more detailed function info
  - Maps machine code address to function name, and possibly function type
  - For stack traces, expression evaluation
- Unwind info
  - Encodes how to restore the registers of the previous call frame
  - For stack traces, unwinding, exception handling
- Source location info (line info)
  - Maps machine code address to source file + line number (and column)
  - For source view, breakpoints, stepping, stack traces
- Variable info
  - Maps machine code address to a scope inside a function
  - With information about the variables in that scope, and where they are stored
  - Allows inspecting variables, and possibly changing their values
- Type info
  - E.g. if a variable is a pointer: to what, and what does that type (e.g. a class) contain
  - Allows inspecting variables of non-trivial type

Debug info is often an order of magnitude larger than optimized executables.

DWARF



- Linux, Mac, etc. use the DWARF Debugging Information Format
- Open standard: https://dwarfstd.org/
- From the early 90s
- Tools for inspection: dwarfdump / objdump -W / readelf -w
- Open-source implementations in compilers and debuggers

But Windows doesn’t (typically) use DWARF…

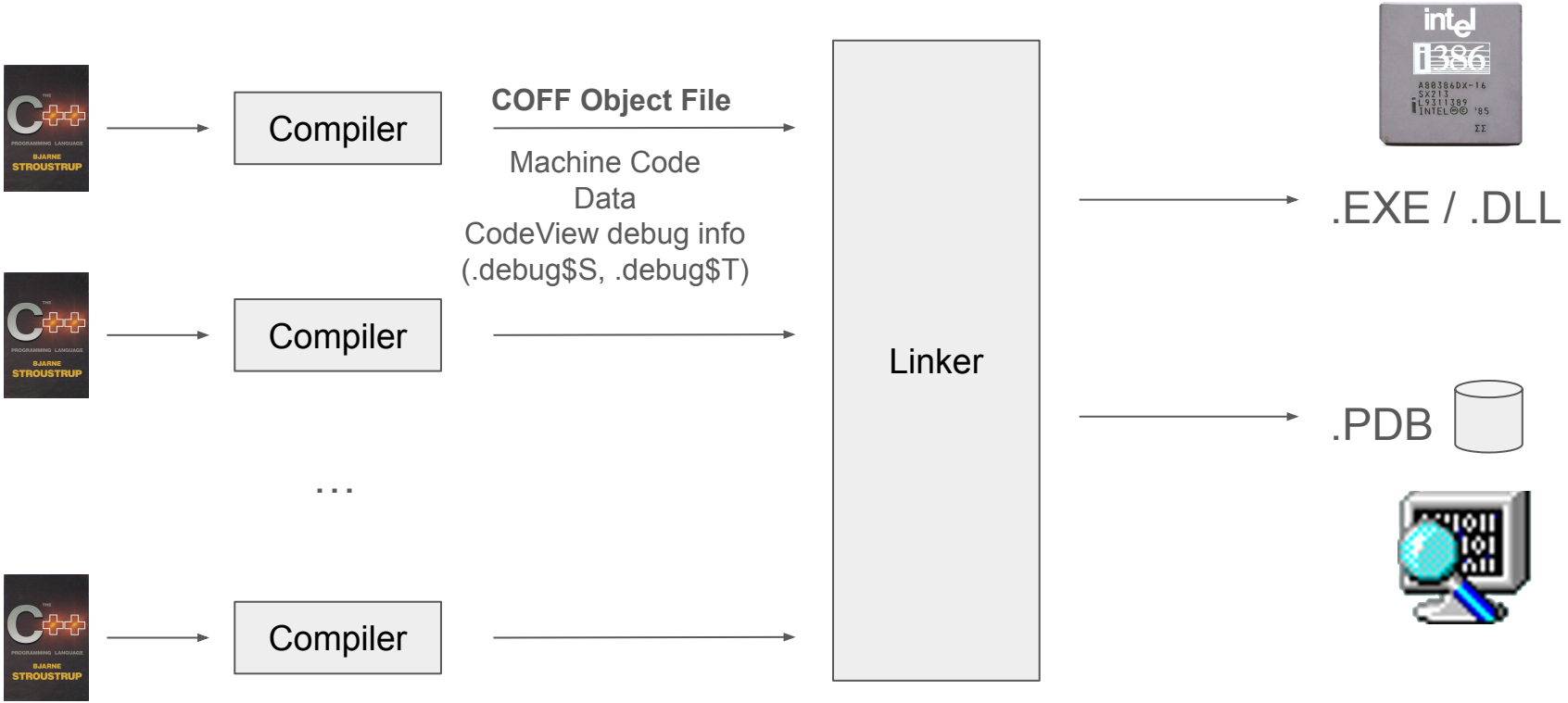
CodeView and PDB

- CodeView: A debugger from the 80s (DOS era, successor to DEBUG.COM and SYMDEB)
- The debugger is long gone, but the format lives on
- MSDN Library had some documentation on this format in the 90s
- (some docs call it “Microsoft Symbol and Type Information” instead)
- There was some support from non-MS compilers
- LLVM added support for line tables in 2014, for better Win/ASan stack traces

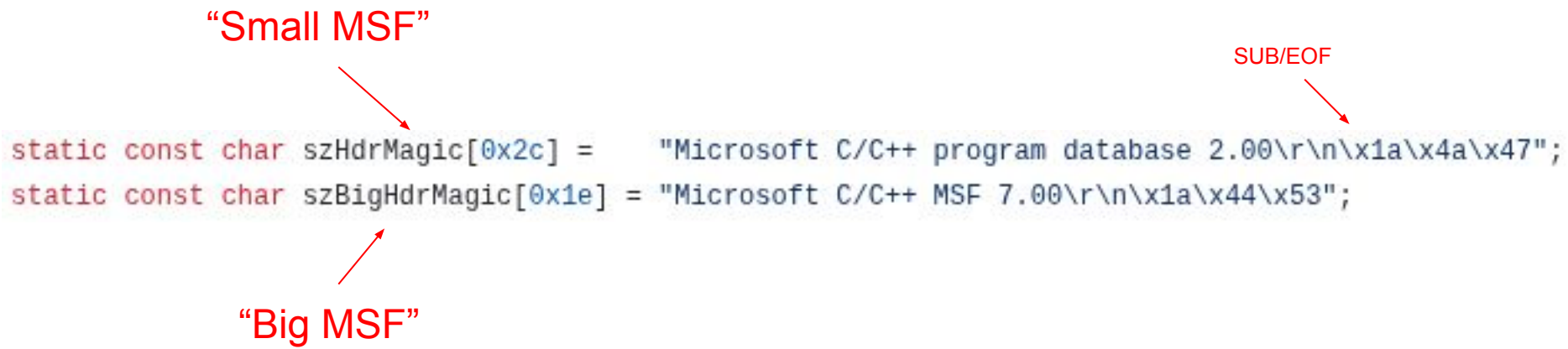
- Program Database (PDB): File format from the 90s – No docs until recently.
- Debug Interface Access SDK (DIA) and DbgHelp for querying debug info (Win only, closed source)

- Microsoft open-sourced parts of PDB and CodeView libraries in 2015 for LLVM’s benefit
- LLVM implemented support and some documentation
- Microsoft released Rust implementation and more docs in 2025/2026

Windows Build Process (/Z7 variant)

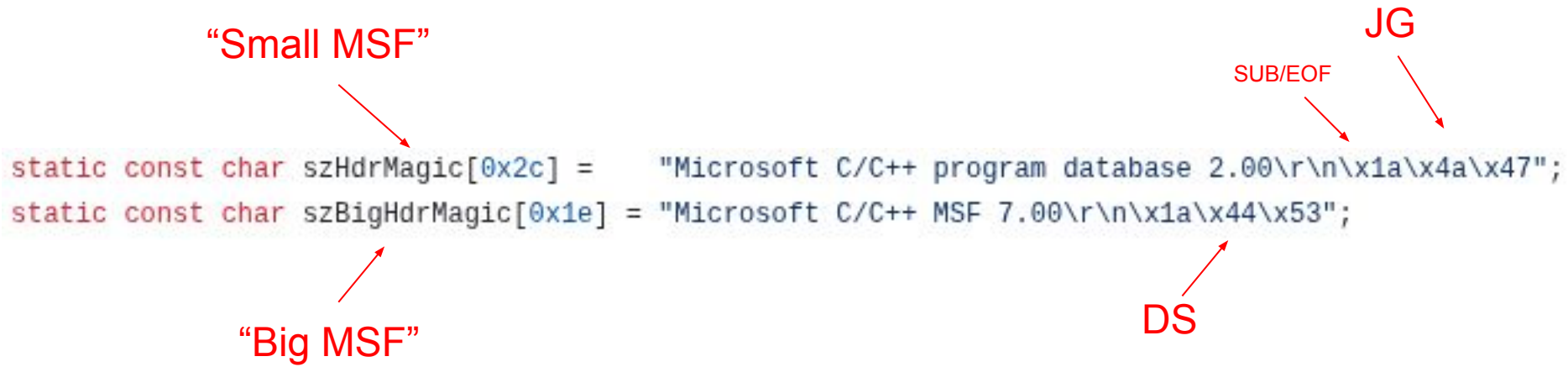


PDB File Header



https://github.com/microsoft/microsoft-pdb/blob/master/PDB/msf/msf.cpp#L961

PDB File Header



https://github.com/microsoft/microsoft-pdb/blob/master/PDB/msf/msf.cpp#L961

The PDB Gang



Left to right: J. Gray, R. Mariani, D. Spalding
“Inside Out: Microsoft—in our own Words”, p. 16

PDB Format Overview

- Multistream File (MSF)
- A set of numbered byte streams
- Like a simple file system
- Transactional updates
- Originally “Program Database” idea, for incremental compilation (late 80s)
- Later adopted in Visual C++ for debug info (“fix the cvpack problem”), etc.
- This talk covers debug info only

Two aspects:

1. *How* data is stored in PDBs (the MSF container format)
2. *What* data is stored (what makes a PDB usable for debugging)

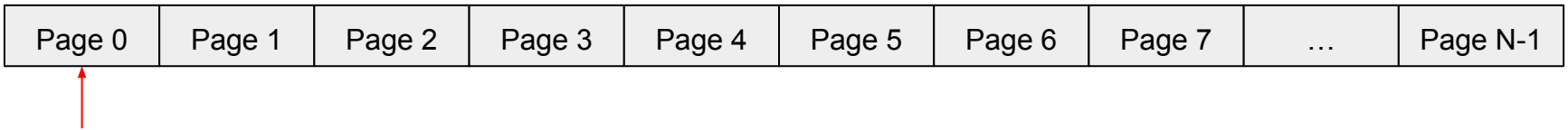
Multistream File (MSF) Structure

- Page-based file format
- Typically 4096-byte pages (can be larger)
- Page numbers are 32-bit (capped to 2^20)
- Stream numbers are 16-bit (max 64 K)
- Stream sizes are 32-bit (max 4 GB)
- .. but many APIs treat it as *signed* 32-bit, so 2 GB (see /PDB4GBSTREAMS)



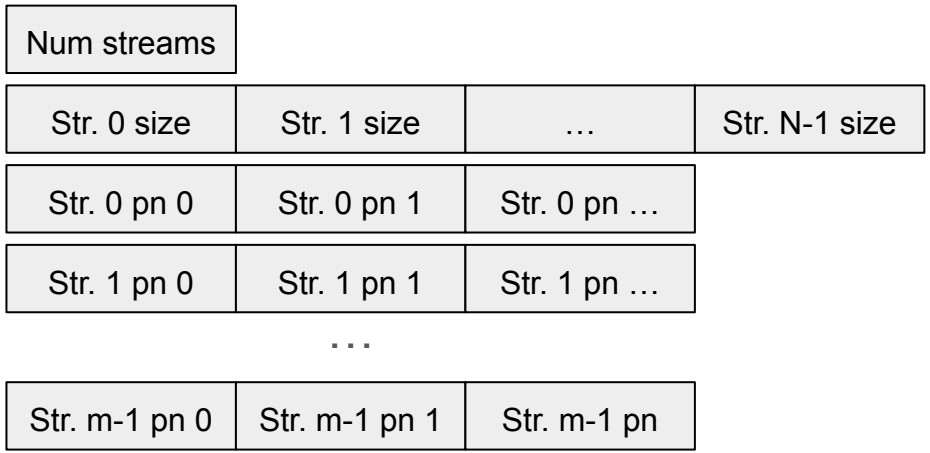
MSF Page 0

- “Header page”
- File magic (signature)
- Page size
- Number of pages
- Free Page Map (FPM) number
- Stream Directory Size
- Stream Directory Page Map



MSF Stream Directory

- 32-bit integers encoding:
- Number of streams
- Size of each stream
- Page numbers of each stream



MSF Stream Directory storage (3 levels)

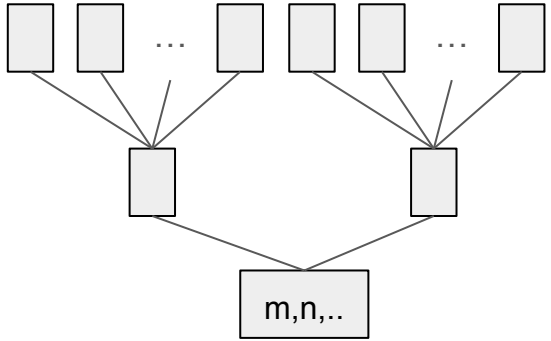
1. Stream Directory
   List of 32-bit integers (stream sizes, page nbrs)
   Stored across one or more pages
2. Stream Directory Page Map
   List of 32-bit page numbers
   Stored across one or more pages
3. Stream Directory Page Map Page Numbers
   Stored after MSF Header on Page 0

MSF Stream Directory storage (3 levels)

Stream Directory

Stream Dir. Page Map

Stream Dir. Pg. Map Pg. Nbr(s)



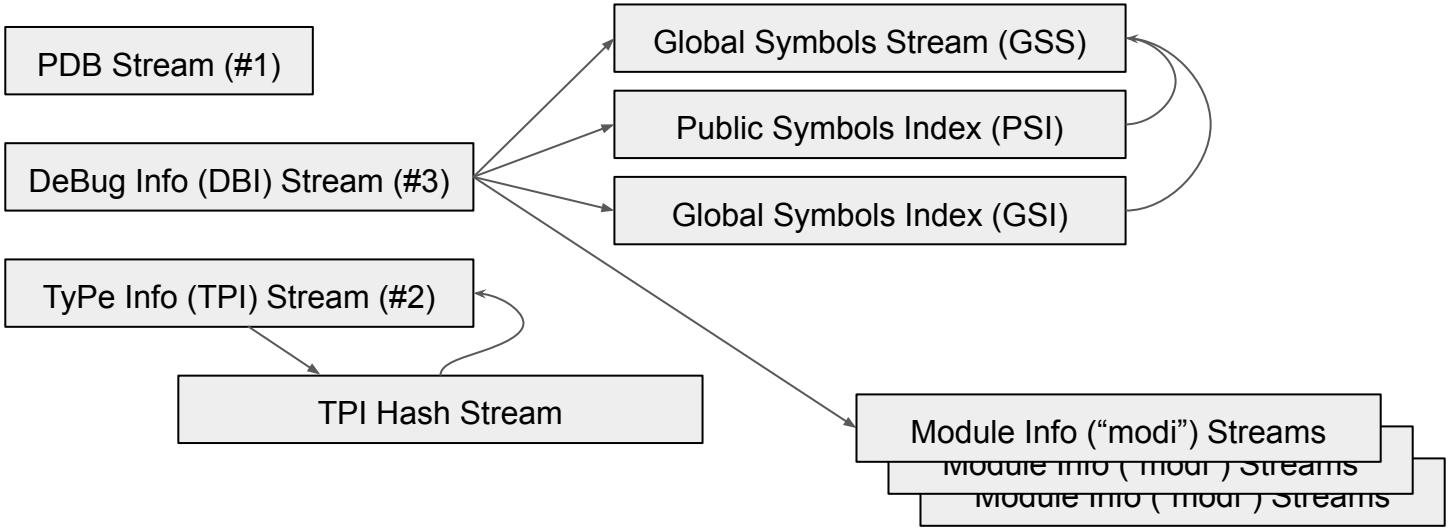
MSF Free Page Map (FPM)

- Bitmap specifying which pages are free (1=free, 0=used)
- There are two FPMs: FPM1 and FPM2
- Active FPM is specified by the header in Page 0
- FPM1 is stored on pages 1 + x * PageSize
- FPM2 is stored on pages 2 + x * PageSize
- (It should have been PageSize / 8)
- This supports transactional updates:
- Start writing new pages (and new page directory) using the inactive FPM
- Commit by flipping active FPM and page directory in the header page

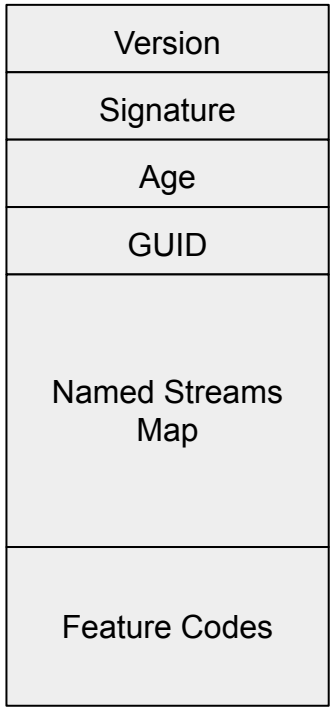
MSF Summary

- Multistream File: Set of numbered byte streams
- Like a filesystem in a file
- Page 0: Header
  - Page size, number of streams, active FPM, Stream Directory location
- Stream Directory
  - Size and page layout of each stream
- Free Page Map (FPM)
  - Which pages are free

PDB Files: What goes in the streams? (simplified)



PDB (Info) Stream (#1)



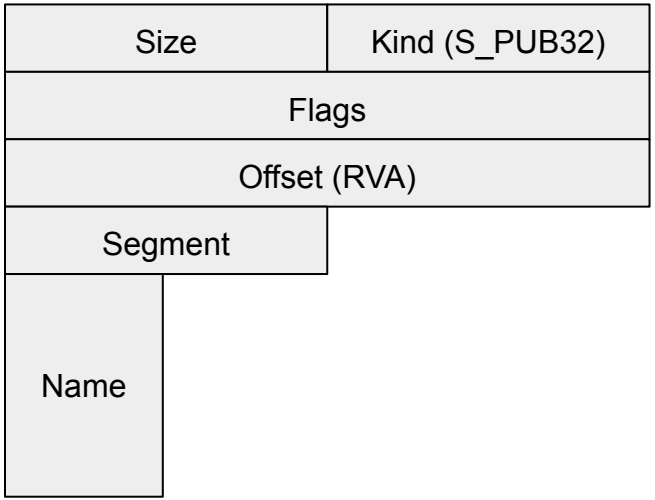
The .EXE/.DLL file references the PDB by filename, Globally Unique ID, and Age.

LLD writes a content hash instead of random GUID

Global Symbols Stream

- Describes global variables, functions, etc.
- Sequence of CodeView “symbol” records

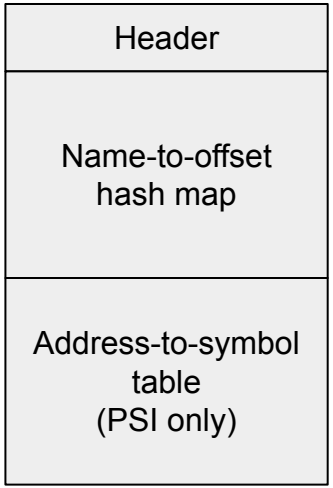
- Example: S_PUB32
- Describes “public” (external linkage) funcs/globals
- Mangled names, address, flags
- Useful for symbolized stack traces
- Retained by /PDBSTRIPPED and pdbcopy -p
- Similar to ELF symbol table



https://github.com/microsoft/microsoft-pdb/blob/master/include/cvinfo.h#L3696
https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/public-and-private-symbols

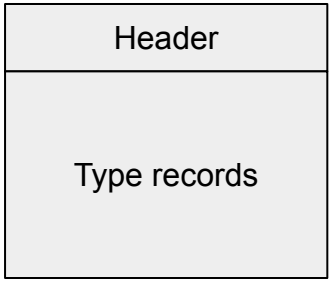
Public / Global Symbols Index (PSI / GSI) Streams

- Indexes into Global Symbols Stream for fast access
- Public Symbols Index: for the S_PUB32 symbols
- Global Symbols Index: for the others
- Hash map from name to symbol offset in GSS
  Allows fast lookup by (exact, case-insensitive) name
- Table of S_PUB32 stream offsets, sorted by symbol address
  Allows binary-search lookup of public symbol by address



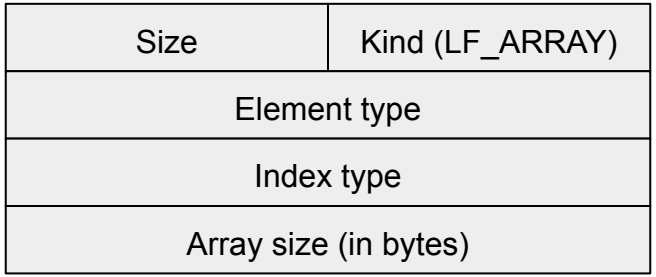
TPI (“TyPe Info?”) Stream (#2)

- A database of all types in the program
  - E.g. pointer to T
  - Or array of T and length N
  - Or class with certain members
  - This can get large, especially in C++
- Header
  - First and last type index (usually 0x1000, 0x1000+N-1)
  - “TPI Hash Stream” number
  - etc.
- Type records
  - CodeView
  - Deduplicated (“hash consing”): one unique record per type
  - In LLVM, the compiler can pre-hash the types



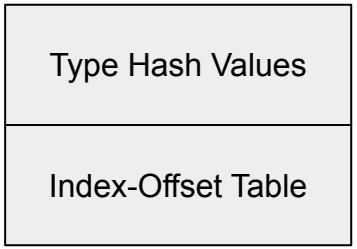
CodeView Type Records

- Each type has a 32-bit ID
- “Leaf” types (int, void, etc.) have pre-assigned IDs below 0x1000
- Type Records have IDs starting at 0x1000 + index in TPI stream
- Example: array type



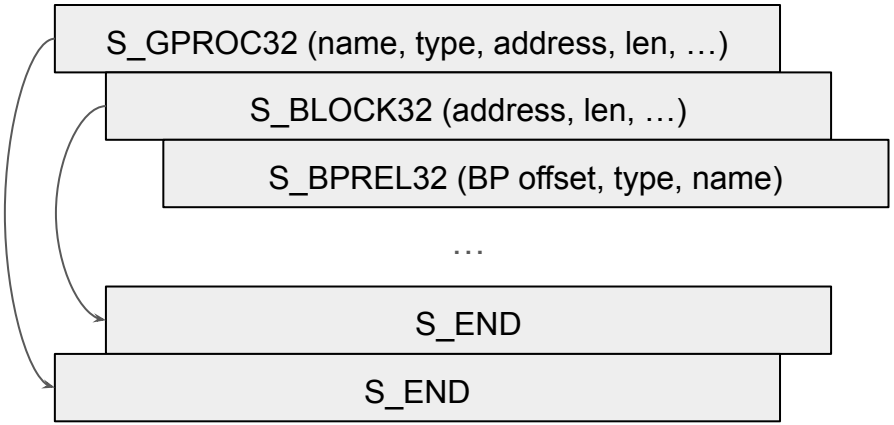
TPI “Hash Stream”

- Stream number in the TPI stream header
- Index-Offset table allows lookup by type index
- Types are deduplicated by hashing
- It seems that hash table is serialized here



Module Info (“modi”) Streams

- Each .obj file generally gets a separate stream
- Symbol records describe the functions and data in that “module”
- For functions, describes the block structure, variables



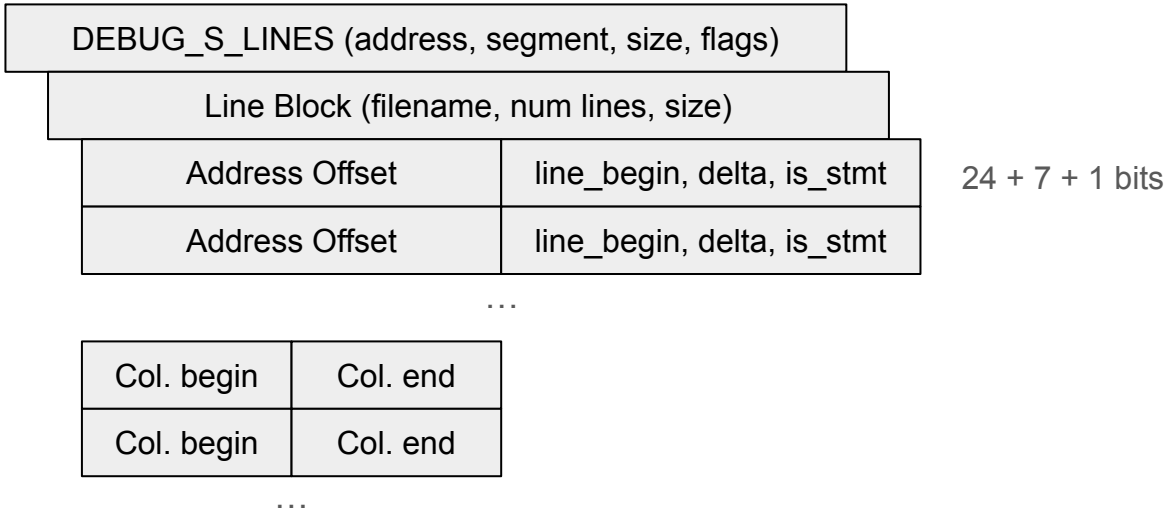
Function

Lexical scope

Frame pointer-relative variable

Module Info (“modi”) Streams continued: Line Info

- Table of filenames and (optional) checksums
  - Filenames are offsets into a string table in a “/names” stream, so deduplicated
- Then a series of “fragments” or “contributions”:



24 + 7 + 1 bits

DBI (“DeBug Info?”) Stream (#3)

- Ties many of the other streams together
- Holds stream nbrs of the globals/publics streams
- Information about each “module” (.obj file)
  - Its module info stream number
  - Its “section contributions” – what address ranges belong to that module
  - What source filenames were used as inputs
  - Optional Frame Pointer Optimization (FPO) data
  - Other optional and/or not well understood data (e.g. for Edit & Continue)

Inspecting PDBs: cvdump.exe

```
C:\src>microsoft-pdb\cvdump\cvdump.exe temp\hello.pdb

Microsoft (R) Debugging Information Dumper  Version 14.00.23611
Copyright (C) Microsoft Corporation.  All rights reserved.
...
*** SYMBOLS

** Module: "C:\src\temp\hello.obj"

(000004) S_OBJNAME: Signature: 00000000, C:\src\temp\hello.obj

(000024) S_COMPILE3:
         Language: C
         Target processor: x64
         Compiled for edit and continue: no
         Compiled without debugging info: no
         Compiled with LTCG: no
         Compiled with /bzalign: no
         Managed code present: no
         Compiled with /GS: yes
         Compiled with /hotpatch: yes
         Converted by CVTCIL: no
         MSIL module: no
         Compiled with /sdl: no
         Compiled with pgo: no
         .EXP module: no
         Pad bits = 0x0000
         Frontend Version: Major = 19, Minor = 29, Build = 30145, QFE = 0
         Backend Version: Major = 19, Minor = 29, Build = 30145, QFE = 0
         Version string: Microsoft (R) Optimizing Compiler

(000060) S_GPROC32: [0001:00006154], Cb: 00000017, Type:            0x1001, main
         Parent: 00000000, End: 000000AC, Next: 00000000
         Debug start: 00000004, Debug end: 00000012
...
```

https://github.com/microsoft/microsoft-pdb/tree/master/cvdump

Inspecting PDBs: llvm-pdbutil

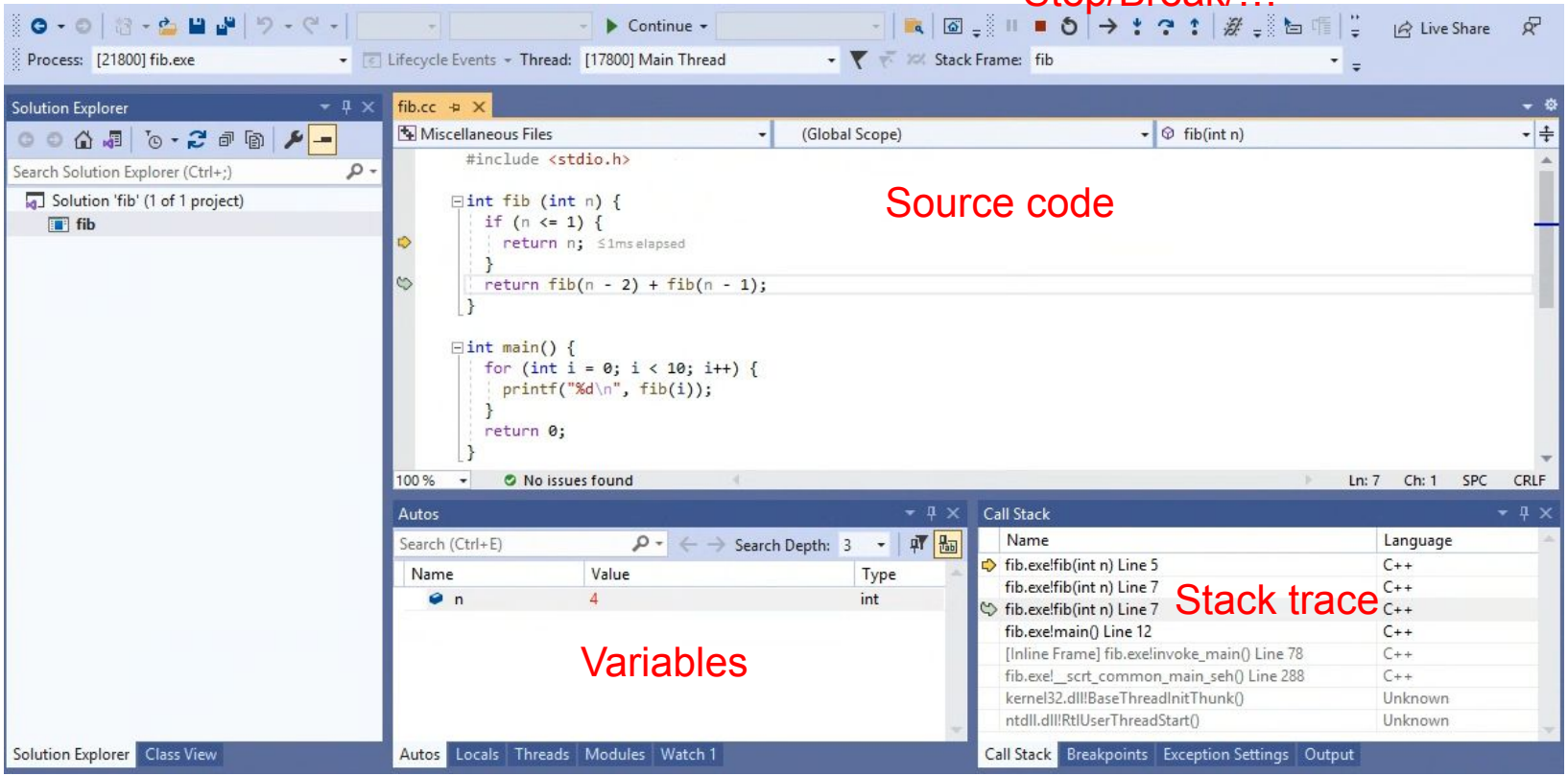
```
C:\src>llvm-pdbutil.exe dump --symbols temp\hello.pdb

                          Symbols
============================================================
  Mod 0000 | `C:\src\temp\hello.obj`:
       4 | S_OBJNAME [size = 32] sig=0, `C:\src\temp\hello.obj`
      36 | S_COMPILE3 [size = 60]
           machine = intel x86-x64, Ver = Microsoft (R) Optimizing Compiler, language = c
           frontend = 19.29.30145.0, backend = 19.29.30145.0
           flags = security checks | hot patchable
      96 | S_GPROC32 [size = 44] `main`
           parent = 0, end = 172, addr = 0001:24916, code size = 23
           type = `0x1001 (int (<no type>))`, debug start = 4, debug end = 18, flags = none

...
```

https://llvm.org/docs/CommandGuide/llvm-pdbutil.html

Recap


Step/Break/…
Continue
Live Share
Process: [21800] fib.exe
Lifecycle Events
Thread: [17800] Main Thread
Stack Frame: fib
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'fib' (1 of 1 project)
fib
Solution Explorer
Class View
fib.cc
Miscellaneous Files
(Global Scope)
fib(int n)
#include <stdio.h>
int fib (int n) {
if (n <= 1) {
return n; ≤1ms elapsed
}
return fib(n - 2) + fib(n - 1);
}
int main() {
for (int i = 0; i < 10; i++) {
printf("%d\n", fib(i));
}
return 0;
}
Source code
100 %
No issues found
Ln: 7
Ch: 1
SPC
CRLF
Autos
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
n
4
int
Variables
Autos
Locals
Threads
Modules
Watch 1
Call Stack
Name
Language
fib.exe!fib(int n) Line 5
C++
fib.exe!fib(int n) Line 7
C++
fib.exe!fib(int n) Line 7
C++
fib.exe!main() Line 12
C++
[Inline Frame] fib.exe!invoke_main() Line 78
C++
fib.exe!__scrt_common_main_seh() Line 288
C++
kernel32.dll!BaseThreadInitThunk()
Unknown
ntdll.dll!RtlUserThreadStart()
Unknown
Stack trace
Call Stack
Breakpoints
Exception Settings
Output

Questions?

References

- https://github.com/microsoft/microsoft-pdb
- https://llvm.org/docs/PDB/index.html
- https://github.com/microsoft/pdb-rs/blob/main/docs/index.md
- https://x.com/jangray/status/6599169977115443320
- https://learn.microsoft.com/en-us/shows/c9-goingnative/goingnative-15-vc-turns-20-ale-turns-40-c9-turns-9
- https://llvm.org/devmtg/2016-11/Slides/Kleckner-CodeViewInLLVM.pdf