The .NET Profiling API
The .NET Profiler API is available since CLR/.NET Framework 1.0
A Profiler depends on the CLR – and not on the .NET Framework

Notable Features
- Assembly loading and unloading events
- Just-in-time (JIT) compilation and code-pitching events
- ReJIT
- Thread creation and destruction events
- Function entry and exit events
- Exceptions
- Transitions between managed and unmanaged code execution
- Information about runtime suspensions
**PROFILING ARCHITECTURE**

- **Program**
  - The .NET application to monitor

- **CLR**
  - Required to execute Program
  - Loads Profiler DLL

- **Profiler DLL**
  - Unmanaged
  - Loaded by CLR into target process

- **IPC mechanism**
  - Interface between Profiler DLL and UI

- **Profiler user interface**
  - Performs costly operations
  - May be a managed application

![Diagram of profiling architecture]
STARTUP

**Startup**
- Process gets created
- CLR is loaded
- CLR loads Profiler DLL
- Profiler subscribes to events

**Execute**
- Application executes
- Profiler receives events
- Profiler communicates with Profiler user interface

**Shutdown**
- CLR informs Profiler of shutdown/detach
- Profiler DLL is unloaded
- Process terminates
A (VERY) BRIEF INTRODUCTION TO COM

Common Object Model (COM)

- Platform and language independent system
- Allows components to locate and communicate with each other
- Based on classes and interfaces
- Each class and interface has a GUID (called CLID or IID)
- COM servers
  - Implemented as DLLs exporting specific functions
  - Register supported CLIDs in the windows registry
- COM clients
  - Request implementations via CLID
  - Request specific interfaces from a class via IID
How does the CLR know if and which profiler DLL to load?

- **Environment Variables**
  - COR_ENABLE_PROFILING=1
    - Must be set to 1 to enable profiling
  - COR_PROFILER_PATH_32=full path to the profiler DLL
  - COR_PROFILER_PATH_64=full path to the profiler DLL
  - COR_PROFILER_PATH=full path to the profiler DLL
    - If present, takes precedence over COR_PROFILER even if invalid
  - COR_PROFILER={CLSID of profiler}
    - The GUID of the COM class implementing ICorProfilerCallback
    - Must be present even if COR_PROFILER_PATH* is used

Prefix CORECLR_ is also allowed

It’s also possible to attach a profiler after application startup (with restrictions)
IDENTIFYING TYPES AND FUNCTIONS

ID: Generated at runtime, typically passed to callbacks

Token: Generated at compile time
Different approaches possible

- Using Enter/Leave/Tailcall hooks
  - Profiler API inserts hook code when method is JITed
  - Hooks must be implemented naked/in assembler
  - Hooks can be installed selectively
  - Hooks can be activated/deactivated during execution

- Instrumenting methods by rewriting IL code
  - Profiler modifies IL code when method is JITed
  - ReJIT feature allows profiler to add/remove instrumentation as required

- Sampling
  - A periodic event (e.g. timer) is used to capture call stacks of threads
  - Prone to deadlocks and race conditions (as one thread suspends another)
MyMethod
{
    Prolog
    Enter Hook
    Body
    Leave Hook
    Epilog
}

ENTER/LEAVE/TAILCALL HOOKS

C++ Class
Profiler::OnEnter

C++ Function
OnEnter

Assembler
EnterNaked

CLR

Prolog
Enter Hook
Body
Leave Hook
Epilog
ANATOMY OF A (MODIFIED) FUNCTION BODY

Original Function Body

- Header
- IL Code
- Exception Handling

Modified Function Body

- Header
- New
- IL Code
- New
- Exception Handling

- New code is inserted
- Original IL code must be moved
- Header must be adjusted
- Exception Handling must be adjusted
ANATOMY OF A (MODIFIED) FUNCTION BODY

- Replace only opcodes
- Nothing else to do ;)
### C# Code

```csharp
public void MyMethod(int value)
{
    if (value == 0) // if (value != 0)
    {
        Console.WriteLine("{0} == 0", value);
    }
    else
    {
        Console.WriteLine("{0} != 0", value);
    }
}
```

### IL Code

```assembly
nop 00
ldarg.1 03
ldc.i4.0 16
ceq FE 01 <- cgt.un FE 03
stloc.0 0A
ldloc.0 06
brfalse.s 2C 15
nop 00
nop 00
br.s 2B 13
nop 00
ldstr 72 19 00 00 70
ldarg.1 03
box 8C 2E 00 00 01
call 28 2A 00 00 0A
nop 00
nop 00
ret 2A
```

### Fat Header

```
13 30   Flags & Size
02 00   MaxStack
32 00 00 00 CodeSize
04 00 00 11 LocalVarSigTok
```
allows the profiler to, well, re-JIT compile method bodys

- Profiler may request to re-JIT a method during execution of the application
- In the callback the profiler modifies the IL body
- New body is used next time when method is executed
- Profiler may request to revert the IL body to its original state

Comes with limitations

- No managed Debugging
- Can not be used with NGEN images
- Not that easy when methods are inlined
- Profiler must be attached at startup
• Profiling (Unmanaged API Reference)

• David Broman's CLR Profiling API Blog
  https://blogs.msdn.microsoft.com/davbr/

• .NET Core runtime GitHub project (CoreCLR)
  https://github.com/dotnet/coreclr/blob/master/src/vm/profilinghelper.cpp

• Rewrite MSIL Code on the Fly with the .NET Framework Profiling API
  MSDN Magazine September 2003, Aleksandr Mikunov

• CLR Profiler
  https://clrprofiler.codeplex.com/

Images: