

.NET (Core) Profiler Workshop

Josef Biehler

Über mich

- biehler-josef.de
- kack.dev
- github.com/gabbersepp
- dev.to/gabbersepp
- gabbersepp@gmail.com

Agenda

- Aufbau des Startprojekts (damit man weiß wie es geht)
- Basiswissen
- Einen Profiler debuggen
- Mehr Informationen rausholen
- FunctionEnter/FunctionLeave 32Bit
- FunctionEnter/FunctionLeave 64Bit
- Funktionsparameter auslesen

Startprojekt

- C++ in VS integrieren



Startprojekt

- Neues Projekt **ATL** mit Standardeinstellungen
- Name: ProfilerWorkshop

The screenshot shows the 'ATL-Projekt' dialog box. The 'Anwendungstyp' dropdown is set to 'Dynamic Link Library (DLL-Datei)'. The 'Dateierweiterung' field is empty. Under 'Dateityp-Handleroptionen', three checkboxes are present: 'Vorschauhandler', 'Miniaturansichtshandler', and 'Suchhandler', all of which are unchecked. The 'Dokumentklassenname' field is empty. Under 'Unterstützungsoptionen', four checkboxes are present: 'Zusammenführen von Proxy-/Stubcode zulassen', 'MFC unterstützen', 'COM+ 1.0-Unterstützung', and 'Unterstützung für Komponentenregistrierung', all of which are unchecked. The 'ATL-basiertes Dokument generieren' checkbox is also unchecked. The 'Dokumentklassen-Headerdateiname' and 'Dokumentimplementierungs-Dateiname' fields are empty, each with a browse button ('...'). The 'Ansichtsklassenname' and 'Ansichtsklassen-Headerdateiname' fields are also empty, each with a browse button ('...'). At the bottom, there are 'OK' and 'Abbrechen' buttons.

ATL-Projekt

Anwendungstyp
Dynamic Link Library (DLL-Datei)

Dateierweiterung
[Empty text box]

Dateityp-Handleroptionen:
☐ Vorschauhandler
☐ Miniaturansichtshandler
☐ Suchhandler

Dokumentklassenname
[Empty text box]

Dokumentklassen-Headerdateiname
[Empty text box] ...

Ansichtsklassenname
[Empty text box]

Unterstützungsoptionen:
☐ Zusammenführen von Proxy-/Stubcode zulassen
☐ MFC unterstützen
☐ COM+ 1.0-Unterstützung
☐ Unterstützung für Komponentenregistrierung

☐ ATL-basiertes Dokument generieren

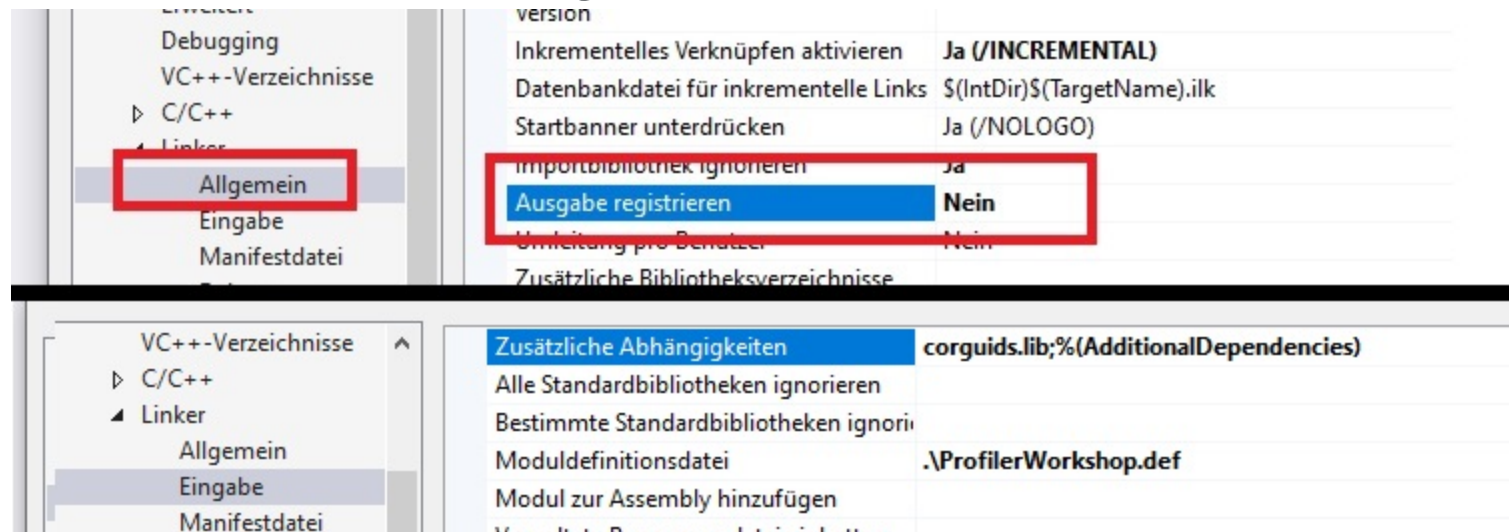
Dokumentimplementierungs-Dateiname
[Empty text box] ...

Ansichtsklassen-Headerdateiname
[Empty text box] ...

OK Abbrechen

Startprojekt

- ProfilerWorkshopPS Projekt: Keine Ahnung für was das benutzt wird. Einfach löschen
- Kein automatisches Registrieren der DLL
- Zusätzliche Abhängigkeiten
- muss für x86 und x64 gemacht werden!



Startprojekt

Änderungen an Dateien:

- **framework.h:** Füge `using namespace ATL;` am Ende der Datei hinzu
- **Resource.h:** Füge `#define IDR_PROFILER 102` ans Ende hinzu, `NewLine` nicht vergessen!
- **ProfilerWorkshop.cpp:** Lösche die Funktion `STDAPI DllInstall(BOOL bInstall, _In_opt_ LPCWSTR pszCmdLine)`.
- **ProfilerWorkshop.def:** Ersetze `LIBRARY` durch `LIBRARY "ProfilerWorkshop.dll"` und entferne die Zeile `DllInstall PRIVATE`

Startprojekt

Hinzufügen neuer Dateien:

- `ProfilerWorkshop.idl` (Ersetzt die vorherige komplett) Link:
- `ProfilerCallback.h` Link:
- jetzt einmal kompilieren, es sollten keine Fehler mehr auftauchen
- `ProfilerCallback.cpp` anlegen. Link:
- Nochmal kompilieren :-)

Startprojekt

- Anlegen eines Testprogramms
- Konsolenapplikation (dotnet core), Version sollte egal sein
- Name "TestApp"

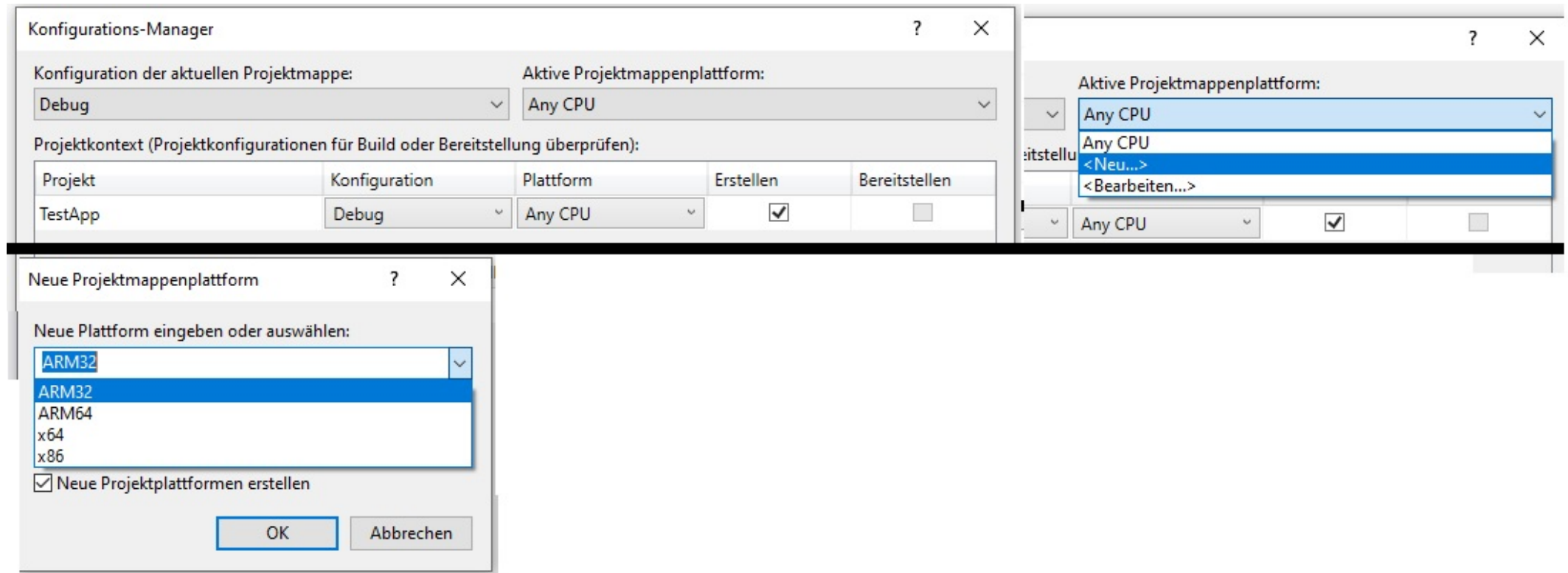
```
namespace TestApp
{
    public class Program
    {
        public static void Main()
        {
            Console.WriteLine("Hello, World!");
            Console.ReadKey();
        }
    }
}
```

Startprojekt

- Any Cpu:

anycpu (default) compiles your assembly to run on any platform. Your application runs as a 64-bit process whenever possible and falls back to 32-bit when only that mode is available.

- Ungünstig! Deswegen -> Definition von festen Targets in TestApp



Startprojekt

Profiler muss der TestApp bzw dotnet bekannt gemacht werden:

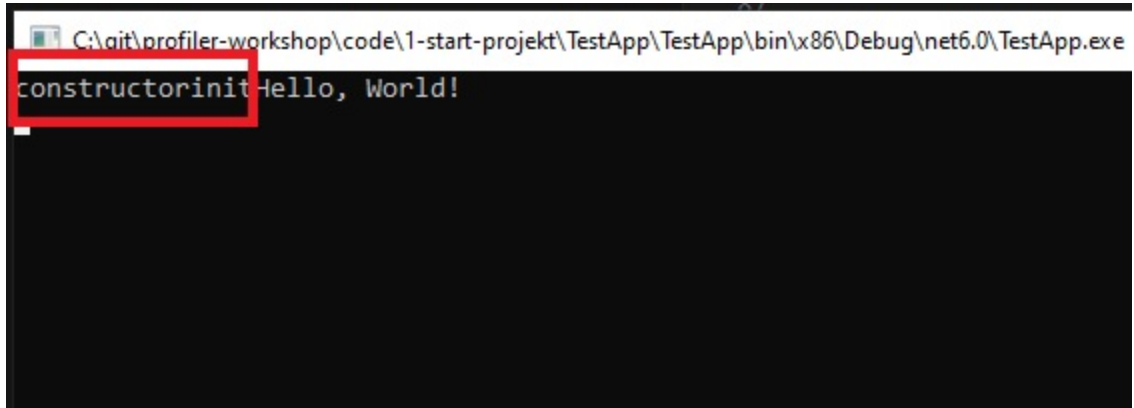
```
SET CORECLR_ENABLE_PROFILING=1
SET CORECLR_PROFILER={b45048d5-6f44-4fbe-ae88-b468a5e4927a}
SET CORECLR_PROFILER_PATH=ProfilerWorkshop/Debug/ProfilerWorkshop.dll
SET COMPLUS_ProfAPI_ProfilerCompatibilitySetting=EnableV2Profiler

START TestApp/TestApp/bin/x86/Debug/net6.0/TestApp.exe
```

Wichtig: Pfade anpassen, je nach `bitness` .

Und: Vorgehen für .Net Framework identisch, außer ENV Variablen

Startprojekt



A screenshot of a Windows command prompt window. The title bar at the top reads "C:\git\profiler-workshop\code\1-start-projekt\TestApp\TestApp\bin\x86\Debug\net6.0\TestApp.exe". The command prompt shows the text "constructorinitHello, World!" on the first line. A red rectangular box highlights the word "constructorinit" at the beginning of the line. The rest of the window is black.

```
C:\git\profiler-workshop\code\1-start-projekt\TestApp\TestApp\bin\x86\Debug\net6.0\TestApp.exe  
constructorinitHello, World!
```

Basiswissen

- Profilercallbacks (z.b. FunctionEnter) werden durch Thread aus Applikation aufgerufen
- Mehrere Threads in App -> parallele Aufrufe in Profiler -> Zugriffe müssen sicher sein!
- **ICorProfilerCallback**: Callbacks werden durch App CLR aufgerufen
- Lifecycle **Initialize()**: Konfigurieren des Profilers

Basiswissen - Initialize()

- Instanz vom Typ `ICorProfilerInfo` anfordern
- Per Flag die Events konfigurieren
- Hooks installieren

Basiswissen - Änderung an ProfilerWorkshop.cpp

- `#include <corprof.h>`
- `CComQIPtr<ICorProfilerInfo2> iCorProfilerInfo;`
- `pICorProfilerInfoUnk->QueryInterface(IID_ICorProfilerInfo2,
(LPVOID*)&iCorProfilerInfo);`

Flag Beispiel: Ausgeben von Exceptions

- `iCorProfilerInfo->SetEventMask(COR_PRF_MONITOR_EXCEPTIONS);`

Im ExceptionThrown Callback:

- `std::cout << "from profiler: \t\t\texception thrown\r\n";`

In TestApp:

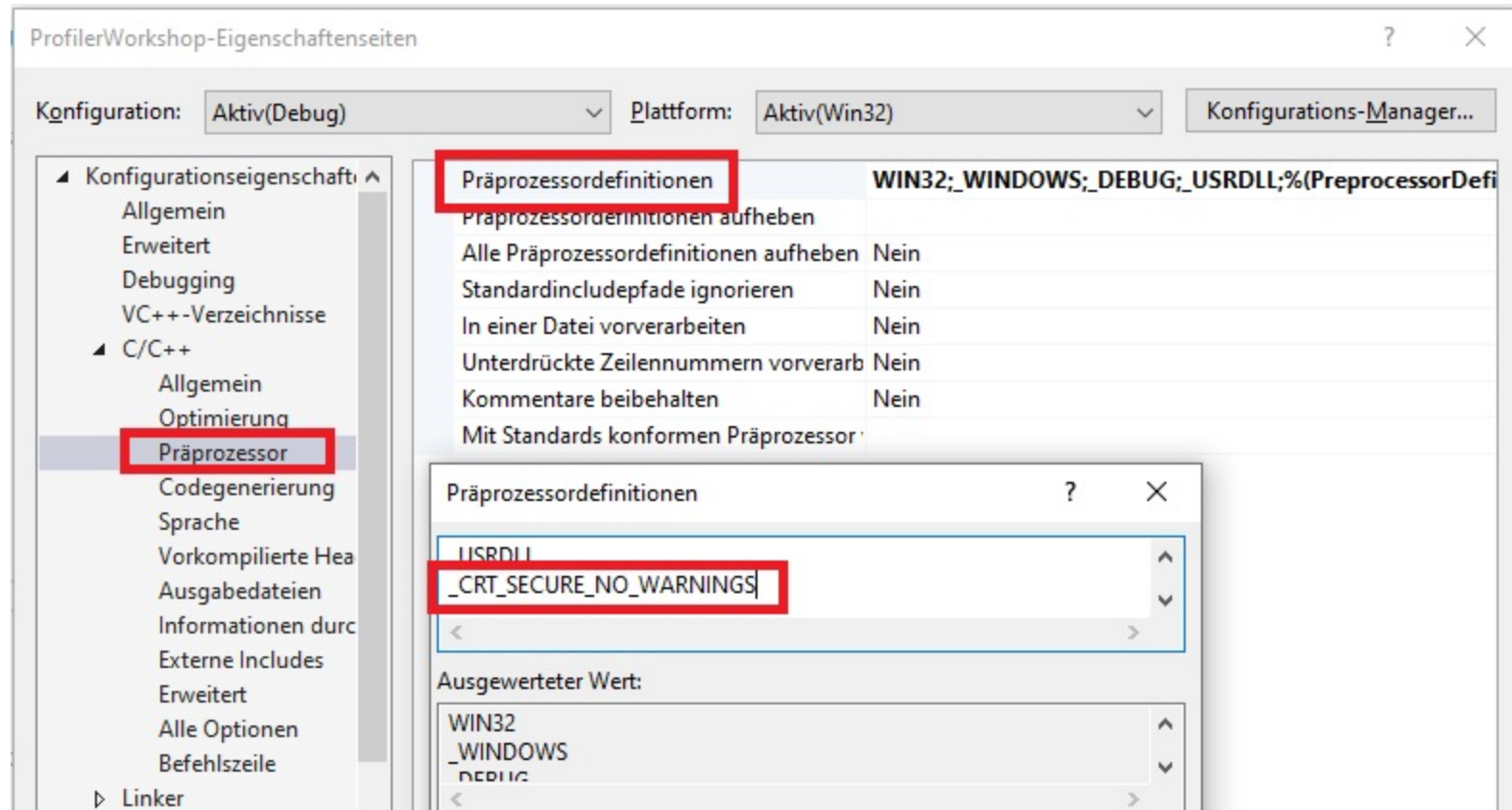
- `throw new Exception();`

Profiler debuggen

- TestApp mit start.bat starten
- VS Debugger attachen (nativen Code)
- Breakpoints setzen

Mehr Informationen rausholen

- Callbacks liefern nur IDs, da performanter
- Weitere Infos durch Methodenaufrufe
- `_CRT_SECURE_NO_WARNINGS` um `wcstombs()` nutzen zu können
- für x86 und x64!



Mehr Infos - Bsp GetClassNameByObjectId

-> Livecoding mit paar Hilfestellungen ;-P

```
iCorProfilerInfo->GetClassFromObject  
iCorProfilerInfo->GetClassIDInfo  
iCorProfilerInfo->GetModuleMetaData  
metadata->GetTypeDefProps  
memset & wcstombs
```

FunctionEnter/Leave 32 Bit

- Flag: `COR_PRF_MONITOR_ENTERLEAVE`
- Werden als Hook installiert
- `ICorProfilerInfo.SetEnterLeaveFunctionHooks2` -> Nur in `Initialize()` möglich!

FunctionEnter/Leave 32 Bit - Wichtig

The `FunctionEnter2` function is a callback; you must implement it. The implementation must use the `__declspec(naked)` storage-class attribute.

The execution engine does not save any registers before calling this function.

On entry, you must save all registers that you use, including those in the floating-point unit (FPU).

On exit, you must restore the stack by popping off all the parameters that were pushed by its caller.

FunctionEnter/Leave 32 Bit - Wichtig

The implementation of `FunctionEnter2` should `not block` because it will delay garbage collection.

The implementation should `not attempt a garbage collection` because the stack may not be in a garbage collection-friendly state.

If a garbage collection is attempted, the runtime will block until `FunctionEnter2` returns.

Also, the `FunctionEnter2` function must `not call into managed code` or in any way cause a managed memory allocation.

FunctionEnter/Leave 32 Bit

Was bedeutet `__declspec(naked)` ?

- Compiler sichert Register am Anfang
- Compiler räumt Stack auf
- Stellt Register wieder her
- Abhängig von Calling Convention!
- All das passiert mit `declspec(naked)` nicht
- Man ist selbst verantwortlich dafür

Exkurs: Calling Convention

- Beschreibt, wie Funktionen aufgerufen werden
- wie werden Parameter übergeben
- wie wird `this` übergeben
- wo landet ein Rückgabewert
- wer kümmert sich um Register und Stack

Exkurs: Calling Convention

- Beispiel `stdcall` (z.b. Win32 API) vs `cdecl` (C default)

Exkurs: Aufrufen von C++ Funktion aus Assembler

- Vereinfacht viele Sachen

FunctionEnter/Leave 32 Bit

Grundkonstrukt:

```
HRESULT __stdcall ProfilerCallback::Initialize(IUnknown* pICorProfilerInfoUnk)
{
    //...
    iCorProfilerInfo->SetEnterLeaveFunctionHooks2((FunctionEnter2*) & FnEnterCallback, (FunctionLeave2*) & FnLeaveCallback, (FunctionTailcall2*) & FnTailcallCallback);
    //...
}2
```

```
void __declspec(naked) FnEnterCallback(
    FunctionID funcId,
    UINT_PTR clientData,
    COR_PRF_FRAME_INFO func,
    COR_PRF_FUNCTION_ARGUMENT_INFO* argumentInfo) {
    __asm {
        ret 16
    }
}

void __declspec(naked) FnLeaveCallback(
    FunctionID funcId,
    UINT_PTR clientData,
    COR_PRF_FRAME_INFO func,
    COR_PRF_FUNCTION_ARGUMENT_INFO* argumentInfo) {
    __asm {
        ret 16
    }
}

void __declspec(naked) FnTailcallCallback(FunctionID funcId,
    UINT_PTR clientData,
    COR_PRF_FRAME_INFO func) {
    __asm {
        ret 8
    }
}
```

FunctionEnter/Leave - Funktionsnamen auslesen

```
bool GetFunctionNameByFunctionId(FunctionID functionId, char* output, ULONG outputLength) {
    IMetaDataImport* metadata;
    mdMethodDef methodToken;
    mdTypeDef typeDefToken;
    wchar_t* functionName = new wchar_t[1000];
    ULONG wcbCount;
    memset(functionName, 0, 1000);

    iCorProfilerInfo->GetTokenAndMetaDataFromFunction(functionId, IID_IMetaDataImport, (IUnknown**)&metadata, &methodToken);
    metadata->GetMethodProps(methodToken, &typeDefToken, functionName, 1000, &wcbCount, NULL, NULL, NULL, NULL, NULL);
    wcstombs(output, functionName, outputLength);
    metadata->Release();
    delete[] functionName;

    return true;
}
```

FunctionEnter/Leave - CPP Funktion aufrufen

```
void __declspec(naked) FnEnterCallback(  
    FunctionID funcId,  
    UINT_PTR clientData,  
    COR_PRF_FRAME_INFO func,  
    COR_PRF_FUNCTION_ARGUMENT_INFO* argumentInfo) {  
    __asm {  
        push dword ptr[ESP + 16]  
        push dword ptr[ESP + 16]  
        push dword ptr[ESP + 16]  
        push dword ptr[ESP + 16]  
        CALL EnterCallbackCpp  
  
        ret 16  
    }  
}
```

FunctionEnter/Leave - Funktionsnamen ausgeben

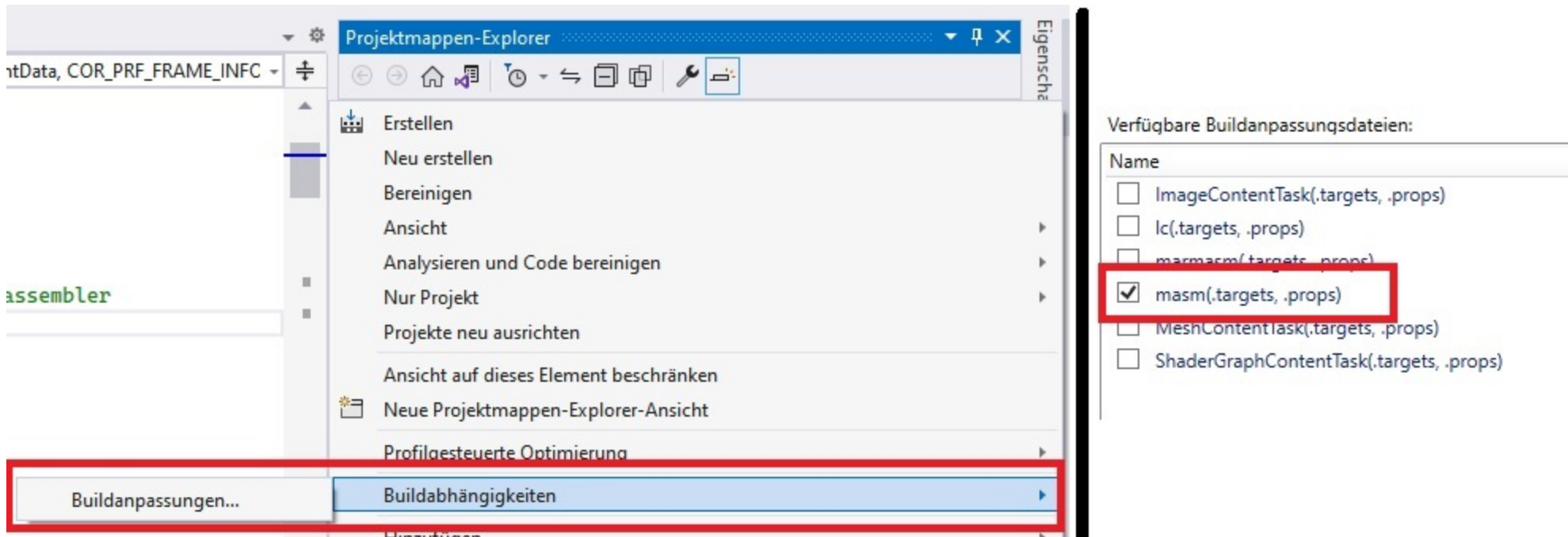
```
void __stdcall EnterCallbackCpp(FunctionID funcId,  
    UINT_PTR clientData,  
    COR_PRF_FRAME_INFO func,  
    COR_PRF_FUNCTION_ARGUMENT_INFO* argumentInfo) {  
    char* output = new char[1000];  
    memset(output, 0, 1000);  
  
    GetFunctionNameByFunctionId(funcId, output, 1000);  
  
    std::cout << "Function enter: " << output << "\r\n";  
  
    delete[] output;  
}
```

FunctionEnter/Leave - 64 Bit

- VS unterstützt bei 64Bit Kompilaten kein inline Assembler
- stattdessen: externe .asm Datei + MASM
- wichtig: `extern c` bei Funktionsdeklaration

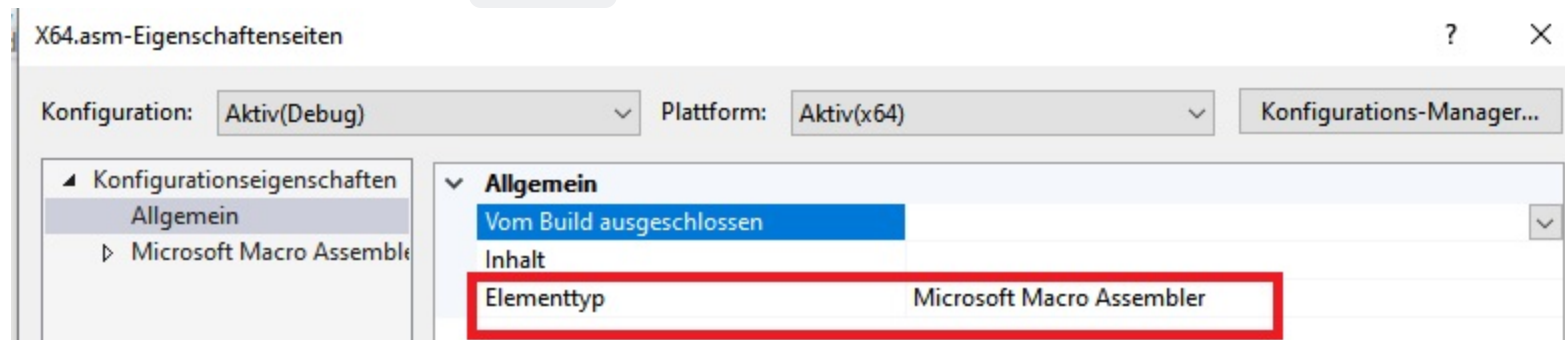
FunctionEnter/Leave - 64 Bit - MASM

- Datei anlegen mit `.asm` Endung
- Projekteinstellungen > Buildabhängigkeiten
- `masm` auswählen



FunctionEnter/Leave - 64 Bit - MASM

- Einstellungen der `.asm` Datei öffnen



FunctionEnter/Leave - 64 Bit - fastcall

```
// just add "extern C" to leave the function name as it is, otherwise i
extern "C" void _stdcall EnterCallbackCpp(FunctionID funcId,
UINT_PTR clientData,
COR_PRF_FRAME_INFO func,
COR_PRF_FUNCTION_ARGUMENT_INFO* argumentInfo) {
    char* output = new char[1000];
    memset(output, 0, 1000);

    GetFunctionNameByFunctionId(funcId, output, 1000);

    std::cout << "Function enter: " << output << "\r\n";

    delete[] output;
}
```

Egal, was hier steht

FunctionEnter/Leave - 64 Bit

```
#ifdef _WIN64
// those functions are defined in the assembler file

EXTERN_C void _fastcall FnEnterCallback(FunctionID funcId,
    UINT_PTR clientData,
    COR_PRF_FRAME_INFO func,
    COR_PRF_FUNCTION_ARGUMENT_INFO* argumentInfo);

EXTERN_C void FnLeaveCallback(FunctionID funcId,
    UINT_PTR clientData,
    COR_PRF_FRAME_INFO func,
    COR_PRF_FUNCTION_ARGUMENT_INFO* argumentInfo);

EXTERN_C void FnTailcallCallback(FunctionID funcId,
    UINT_PTR clientData,
    COR_PRF_FRAME_INFO func);
#else

void __declspec(naked) FnEnterCallback(
    FunctionID funcId,
    UINT_PTR clientData,
    COR_PRF_FRAME_INFO func,
    COR_PRF_FUNCTION_ARGUMENT_INFO* argumentInfo) {
    __asm {
        push dword ptr[ESP + 16]
        push dword ptr[ESP + 16]
        push dword ptr[ESP + 16]
        push dword ptr[ESP + 16]
        CALL EnterCallbackCpp

        ret 16
    }
}
```

Funktionsparameter auslesen

```
public static void Main()
{
    Console.ReadKey();
    Blub_i(1);
    Blub_arr(new []{1,2,3,4,5,6});
    Blub_str("Hello, World");
    Console.ReadKey();
}

public static void Blub_i(int i)
{
    Console.WriteLine(i);
}

public static void Blub_str(string str)
{
    Console.WriteLine(str);
}

public static void Blub_arr(int[] intArray)
{
    Console.WriteLine(intArray.Length);
}
```

Funktionsparameter auslesen - COR_PRF_ENABLE_FUNCTION_ARGS

```
iCorProfilerInfo->SetEventMask(COR_PRF_MONITOR_EXCEPTIONS |  
COR_PRF_MONITOR_ENTERLEAVE | COR_PRF_ENABLE_FUNCTION_ARGS);
```

Funktionsparameter auslesen

- `COR_PRF_FUNCTION_ARGUMENT_INFO*` `argumentInfo`
- struct welche Speicherblöcke mit Parametern beschreibt
- `argumentInfo->numRanges` Anzahl solcher Blöcke
- `argumentInfo->ranges` Array an Daten

In unserem Beispiel einfach: Nur ein Parameter

```
argumentInfo->ranges[0].startAddress
```

Funktionsparameter auslesen - Startaddress

- `startAddress` Bedeutung abhängig von Parametertyp
- `value` Type: Pointer zu Wert
- `object`: Pointer auf Pointer zu Method Table Pointer
- `struct`: Pointer zu struct
- `fastcall` : Parameter in Register, werden extra in Speicher geschoben

Funktionsparameter auslesen - Repräsentation der Daten

- Bücher lesen
- Debuggen
- Siehe Beispiel

Hierzu:

`fastcall` : Parameter landen v.l.n.r in `RCX, RDX, R8, R9`

Funktionsparameter auslesen - Code

Siehe Code