Szoftver technológia

Build systems

Cserép Máté
ELTE Informatikai Kar
2019.

Build systems

C++ programok fordítása

```
g++ -c -o foo.o foo.cpp \
    -O2 -std=c++11 -pedantic -I./include/


... további fordítási egységek ...


g++ -c -o main.o main.cpp \
    -O2 -std=c++11 -pedantic -I./include/
g++ -o program.exe  foo.o ...  main.o
```

- A programok manuális fordítása már kisebb programoknál is könnyen nehezen kezelhetővé válik.
- Nagyobb programoknál nem követhető mely fordítási egységek újrafordítása szükséges.

Build systems

Fordító eszközök kategorizálása

- Önálló (*standalone*)
  - Make, NMake, Scons, Jom, BJam, Ninja
- Fejlesztő környezetebe integrált (*integrated*)
  - Visual Studio, Xcode, Eclipse
- Generátorok (*generators*)
  - CMake, Autotools, GYP

Build systems

Make

- GNU Make
  - https://www.gnu.org/software/make/
- A fordítást célokon (*target*) és szabályokon (*rule*) keresztül definiáljuk.
- A szabályok függőségeket (*dependency*) és utasításokat (*command*) tartalmazhatnak.
- A szabályrendszert az ún. *Makefile* tartalmazza.
- Futtatás: **`make`** vagy **`make -f MyMakefile`**
- Példa Makefile:

```
program: foo.cpp main.cpp
    g++ -o program foo.cpp main.cpp -std=c++11
```

Build systems

Make

- Változók használata:

```
CXX=g++
CFLAGS=-std=c++11 -pedantic

program: foo.cpp main.cpp
	$(CXX) -o program foo.cpp main.cpp $(CFLAGS)
```

Build systems

Make

- Több fordítási egység:

```
CXX=g++
CFLAGS=-std=c++11 -pedantic

foo.o: foo.cpp foo.h
  $(CXX) -c -o foo.o foo.cpp $(CFLAGS)

main.o: main.cpp
  $(CXX) -c -o main.o main.cpp $(CFLAGS)

program: foo.o main.o
  $(CXX) -o program foo.o main.o $(CFLAGS)
```

Build systems

Make

- Automatikus változók:
  - $@: a cél (fájl)neve
  - $<: az első függőség neve
  - $^: az összes függőség neve
  - $?: a célnál frissebb függőségek nevei
  - $(@D): a cél nevének könyvtárrésze
  - $(@F): a cél nevének fájlrésze
  - stb.

Build systems

Make

- Automatikus változók – 1. példa:

```
CXX=g++
CFLAGS=-std=c++11 -pedantic
DEPS=foo.h

%.o: %.c $(DEPS)
	$(CXX) -c -o $@ $< $(CFLAGS)

program: foo.o main.o
	$(CXX) -o program foo.o main.o $(CFLAGS)
```

Build systems

Make

- Automatikus változók – 2. példa:

```
CXX=g++
CFLAGS=-std=c++11 -pedantic
DEPS=foo.h
OBJ=foo.o main.o

%.o: %.c $(DEPS)
	$(CXX) -c -o $@ $< $(CFLAGS)

program: $(OBJ)
	$(CXX) -o $@ $^ $(CFLAGS)
```

Build systems

Make

- Forrás és fordítási könyvtárszerkezet kezelése (pl. *include*, *obj*):

```
IDIR=../include
ODIR=obj
CXX=g++
CFLAGS=-std=c++11 -pedantic -I$(IDIR)
_DEPS=foo.h
DEPS=$(patsubst %, $(IDIR)/%, $(_DEPS))
_OBJ=foo.o main.o
OBJ = $(patsubst %, $(ODIR)/%, $(_OBJ))

$(ODIR)/%.o: %.c $(DEPS)
  $(CXX) -c -o $@ $< $(CFLAGS)
$(ODIR)/program: $(OBJ)
  $(CXX) -o $@ $^ $(CFLAGS)
```

Build systems

Make

- Hamis (*phony*) célok:

```
.PHONY: all
all: $(ODIR)/program

... további célok ...

.PHONY: clean
clean:
  rm -f $(ODIR)/*.o $(ODIR)/program
```

Build systems

Make

- Modulok (alkönyvtárak) kezelése:

```
.PHONY: all

all: program

  $(MAKE) -C module_a
  $(MAKE) -C module_b
  $(CXX) -o program $(CFLAGS) \
    module_a/obj/modul_a.o \
    module_a/obj/modul_b.o
```

Build systems

CMake

- CMake
  - https://cmake.org/
- Platformfüggetlen és fordítófüggetlen generátor alacsonyabb szintű fordító eszközökhöz.
  - Pl. GNU Make, MSVC, Ninja.
- A konfigurációt a *CMakeLists.txt* állományban adjuk meg.
- Futtatás: **`cmake <paraméterek>`**
- Példa CMakeLists.txt:

```
cmake_minimum_required (VERSION 3.7)
project(HelloWorld)
add_executable(HelloWorld main.cpp foo.cpp)
```

Build systems

CMake

- Változók használata:

```
cmake_minimum_required (VERSION 3.7)
project(HelloWorld)

set(SRCS \
  main.cpp \
  foo.cpp \
  foo.h)

add_executable(HelloWorld ${SRCS})
```

Build systems

CMake

- C++ fordító paraméterezése:

```
cmake_minimum_required (VERSION 3.7)
project(HelloWorld)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_FLAGS "-O2 -pedantic")

add_executable(HelloWorld main.cpp foo.cpp)
```

Build systems

CMake

- Statikus vagy dinamikus könyvtár készítése:

```
cmake_minimum_required (VERSION 3.7)
project(MyStack)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_FLAGS "-O2 -pedantic")

add_library (my_stack_static STATIC mystack.cpp)
add_library (my_stack_dynamic SHARED mystack.cpp)
```

Build systems

CMake

- *Include path* bővítése:

```
cmake_minimum_required (VERSION 3.7)
project(HelloWorld)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_FLAGS "-O2 -pedantic")

include_directories("include")

add_executable(HelloWorld main.cpp foo.cpp)
```

Build systems

CMake

- Csomag függőségek definiálása:

```
cmake_minimum_required (VERSION 3.7)
project(MyProgram)

find_package(SomePackage)
include_directories(...)
link_directories(...)
link_libraries(...)
```

- Elegendő lehet a célhoz linkelni:

```
target_link_directories(mytarget ...)
target_link_libraries(mytarget ...)
```

Build systems

CMake

- OpenCV használata:

```
cmake_minimum_required (VERSION 3.7)
project(MyProgram)

find_package(OpenCV REQUIRED)
link_libraries(${OpenCV_LIBS})
```

Build systems

CMake

- Boost használata:

```
cmake_minimum_required (VERSION 3.7)
project(MyProgram)

set(Boost_USE_STATIC_LIBS ON)
set(Boost_USE_STATIC ON)
find_package(Boost REQUIRED
  COMPONENTS filesystem log program_options)
include_directories(${Boost_INCLUDE_DIRS})
link_libraries(${Boost_LIBRARIES})
```

Build systems

CMake

- Qt használata:

```
cmake_minimum_required (VERSION 3.7)
project(QtHelloWorld)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTOUIC ON)

find_package(Qt5Widgets CONFIG REQUIRED)

set(SRCS mainwindow.ui mainwindow.cpp main.cpp )
add_executable(helloworld WIN32 ${SRCS})
target_link_libraries(helloworld Qt5::Widgets)
```

Build systems

CMake

- Modulok (alkönyvtárak) kezelése:

```
cmake_minimum_required (VERSION 3.7)
project(MyProgram)

... Konfiguráció ...

add_subdirectory(module_a)
add_subdirectory(module_b)
```

Build systems

CMake

- Jellemzően egy külön *build könyvtárba* szeretnénk fordítani:

  `mkdir build && cd build`

  `cmake ../` vagy `cmake ../src`

- Több féle generátor közül választhatunk:
  - GNU Makefile használata:
    `cmake -G "Unix Makefiles" ../`
  - Microsoft Visual Studio:
    `cmake -G "Visual Studio 15 2017 Win64" ../`
  - Xcode:
    `cmake -G Xcode ../`
- Majd fordítjuk a programot:
  - Makefile: `make`
  - MSVC: `msbuild MyProgram.sln`

Build systems

CMake

- Megadhatunk egy telepítési könyvtárat, ahova a végső binárisokat át szeretnénk másolni:

```
cmake -DCMAKE_INSTALL_PREFIX=../install ../src
```

- Példa CMakeLists.txt:

```
cmake_minimum_required (VERSION 3.7)
project(MyProgram)

set(SRCS main.cpp foo.cpp foo.h)
add_executable(HelloWorld ${SRCS})
install(TARGETS HelloWorld
  DESTINATION ${CMAKE_INSTALL_PREFIX})
```

- Makefiles: **`make install`**
- MSVC: **`msbuild INSTALL.vcxproj`**

Build systems

MSBuild / XBuild

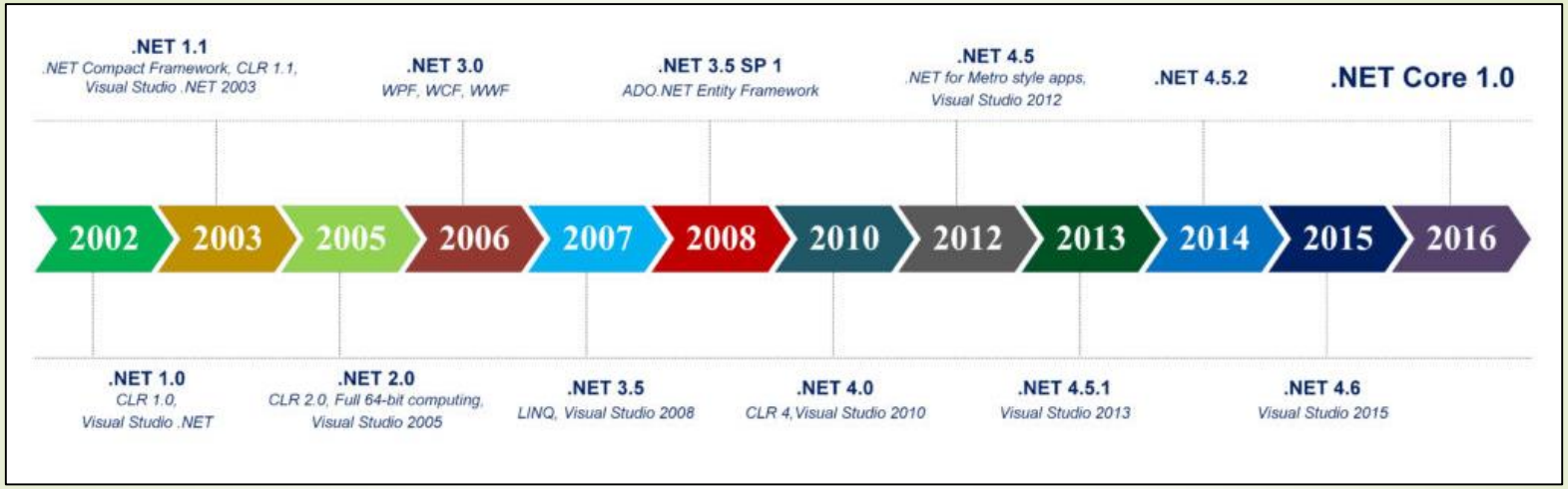
- A Microsoft saját build eszköze Visual Studio projektekre.
  - A *.sln* solution és a *.csproj/.vcxproj* projekt fájlok alapján fordítja le az alkalmazást.
  - A projekt állományok XML formátumúak, a projekthez tartozó fájlok mellett fordítási konfigurációk és egyéb beállítások is megadhatók bennük.
- Például:

```
msbuild SolutionFile.sln /t:Build /p:Configuration=Release
```

  - a *SolutionFile.sln* fájlban adott solutionra
  - a *Build* target végrehajtása
  - a *Release* konfiguráció szerint
- Az MSBuild Mono projekt alatti implementációja az *XBuild*.

Build systems

A .NET Framework keretrendszer



- *Problémák a .NET Framework keretrendszerrel:*
  - Windows-központú megközelítés
  - Monolitikus, nem megfelelően modularizált felépítés
  - Zárt forráskód

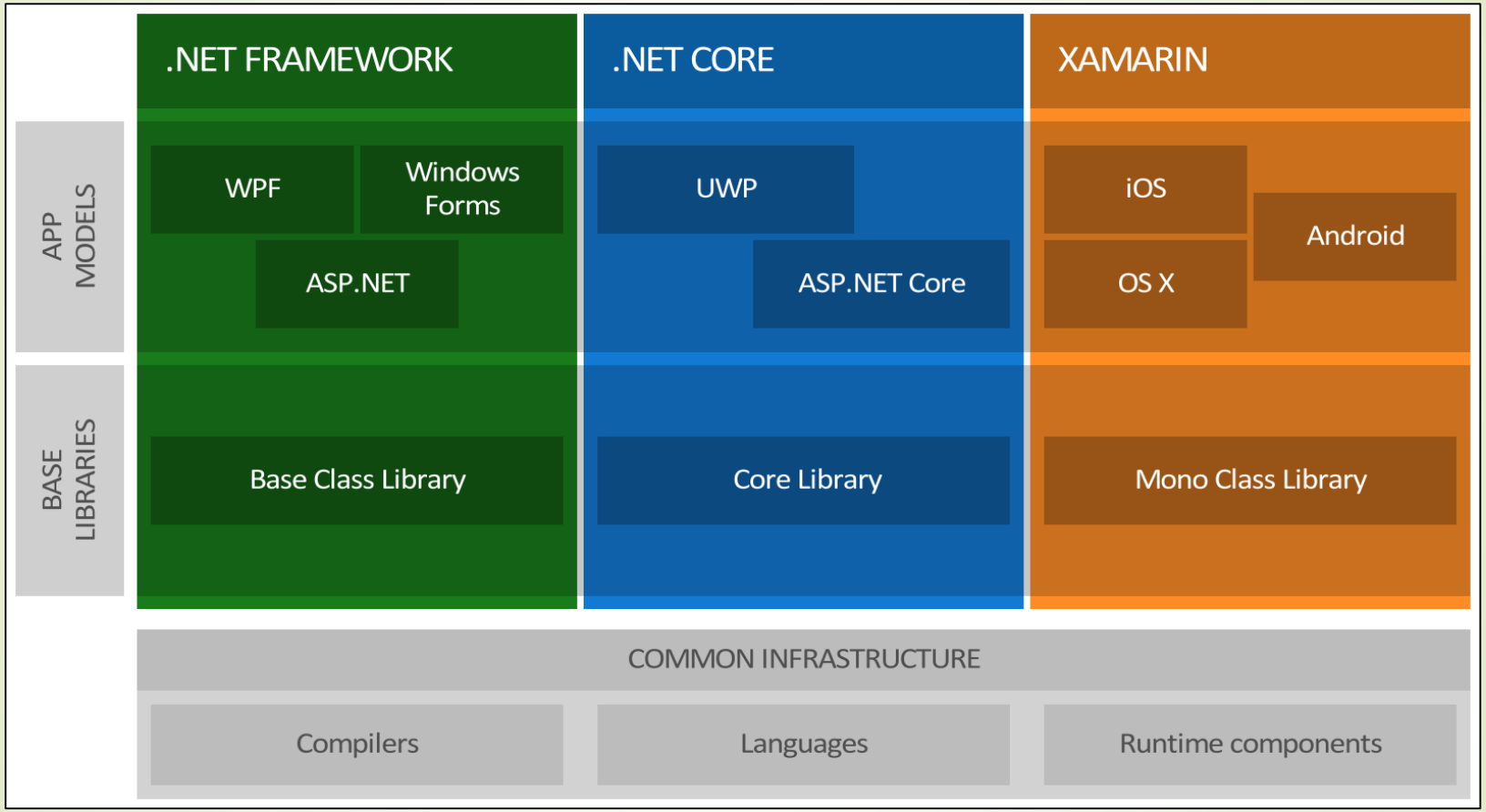
Build systems

A .NET Core keretrendszer

- *A .NET Core ezekre nyújt megoldást:*
  - Cross-platform (Windows, Linux, macOS)
  - Modularizált felépítés, csak az alkalmazáshoz szükséges komponenseknek kell jelen lennie.
  - Nyílt forráskód (https://github.com/dotnet/core)
- *Érdemes .NET Core-t használni:*
  - Platformfüggetlen és/vagy open source projekteknél
  - Grafikus felülettel nem rendelkező alkalmazások esetében, tipikusan ilyenek a szerveralkalmazások
  - Microservicek készítésekor (modularizáltság), konténerek használatakor (pl. Docker)
  - Magas teljesítmény és skálázhatóság esetén (a .NET Core és az ASP.NET Core teljesítménye jelentősen jobb)

Build systems

A .NET Framework és Core kapcsolata



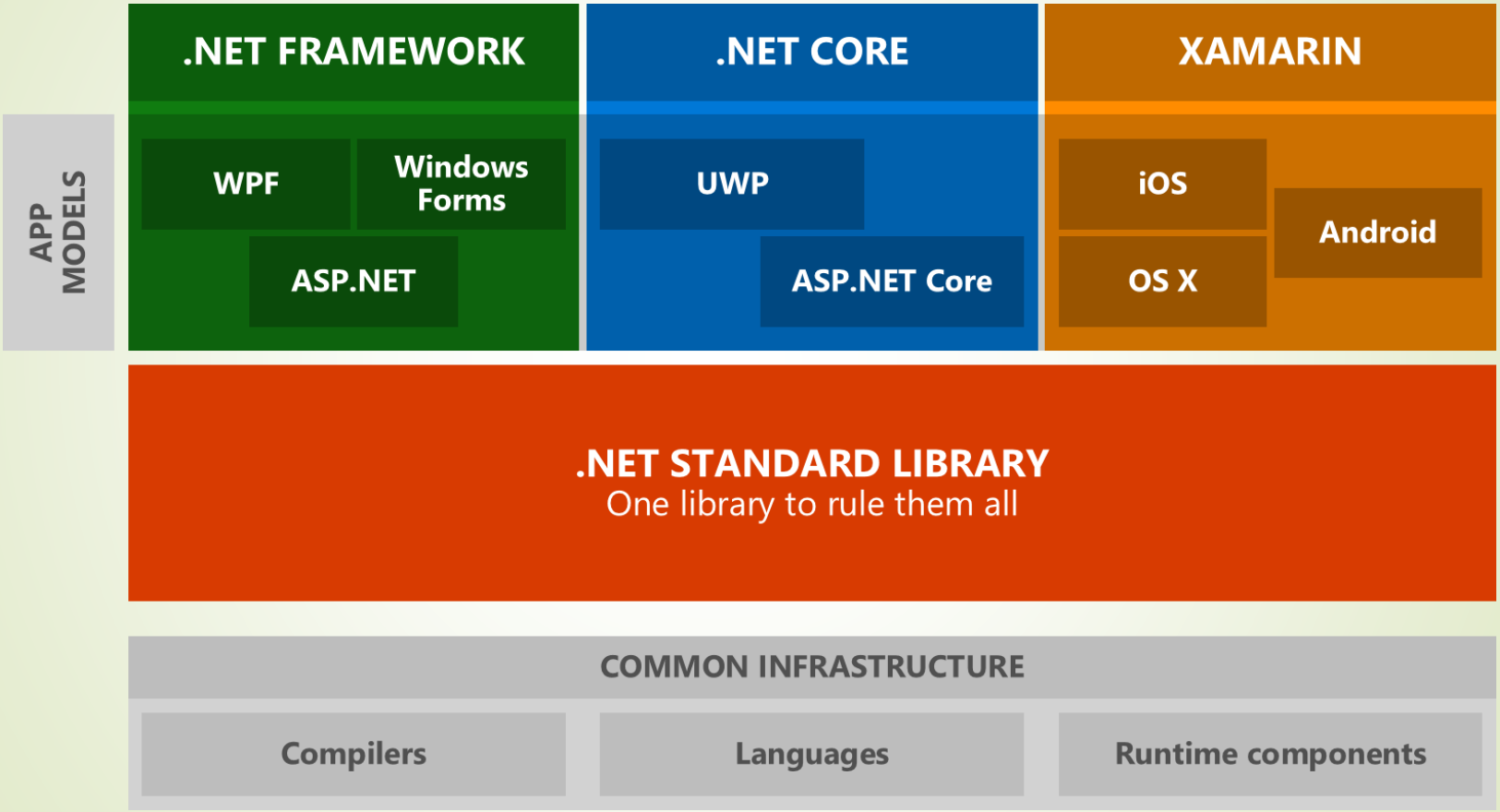
Build systems

A .NET Standard

- *Felmerülő problémák:*
  - Különböző keretrendszerben írt alkalmazások integrálása
  - Általános felhasználható programkönyvtárak fejlesztése
- *.NET Standard:*
  - Közös, megosztott API az egyes keretrendszer BCL-ek (Base Class Library) felett
  - Felváltja a PCL (Portable Class Library) projekteket

Build systems

A .NET Standard



Build systems

A .NET Standard

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework (with .NET Core 1.x SDK)	4.5	4.5	4.5.1	4.6	4.6.1	4.6.2		
.NET Framework (with .NET Core 2.0 SDK)	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299
Windows	8.0	8.0	8.1					
Windows Phone	8.1	8.1	8.1					
Windows Phone Silverlight	8.0							

Build systems

Fordítás .NET Core keretrendszer alatt

- A platformfüggetlen .NET Core keretrendszer alatt a *dotnet* konzolos utasítással érhetjük el az SDK-t.
- A projekteket változatlanul solutionökre (.sln fájl) és projektekre (*.csproj/.vcxproj* fájlok) osztjuk, amelyek XML formátumúak.
- Például:
  - NuGet csomagok visszaállítása: `dotnet nuget restore`
  - Fordítás: `dotnet build path/to/SolutionFile.sln`
    Kurrens könyvtárra: `dotnet build`
  - Fordítás a Release konfiguráció szerint:
    `dotnet build --configuration Release`
  - Futtatatás: `dotnet run path/to/SolutionFile.sln`
    Már előállított binárisra: `dotnet path/to/Binary.exe`