C# Compiler Tidbits

Bart J.F. De Smet
bartde@outlook.com

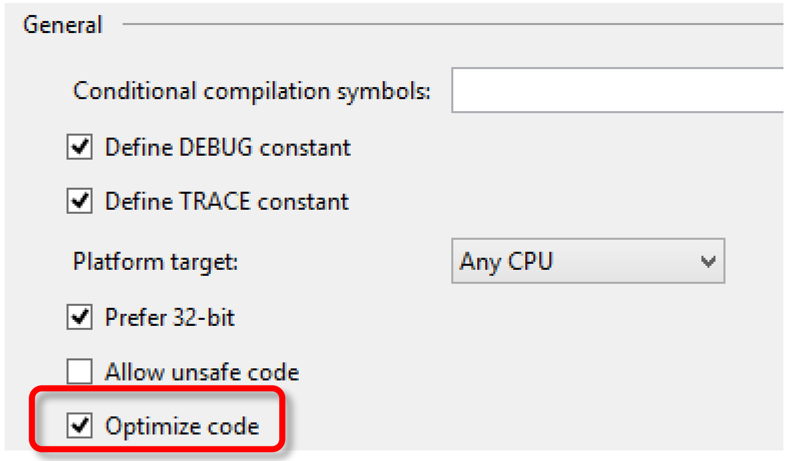
The Command-line Compiler

- **csc.exe**
  - Invoked by MSBuild and Visual Studio
  - Quite a few knobs to turn
    - Not quite as many as C++ ☺
  - Supports response file (.rsp)

- **Compiler options**
  - Specification of target (.dll, .exe, .winmd, etc.)
  - Target platform specification (“Any CPU”, x86, x64, etc.)
  - Referencing of dependencies
  - Code generation options (debug, optimize)
  - Language options (version, checked arithmetic, unsafe code)
  - Security settings (strong name signing, etc.)
  - Advanced options

Compiler Optimizations

- **IL code generation**
  - Default code generation for fragments
    - Stores to local variable slots
    - Branching for control flow
  - Many nop instructions
    - E.g. for curly braces
    - Allows setting breakpoints
- **Compiler /o+ option**
  - Enables another compiler pass
    - Branch optimizations
    - Eliminates empty code blocks
    - Gets rid of unused locals
  - Only basic optimizations
    - JIT does a lot more at runtime
    - NGEN can take its time to optimize
  - Should not change meaning of the code (d'oh)



General

Conditional compilation symbols:

☑ Define DEBUG constant

☑ Define TRACE constant

Platform target: Any CPU

☑ Prefer 32-bit

☐ Allow unsafe code

☑ Optimize code

Compiler Optimizations

- **Excessive use of locals**
  - Compilation of return statement
    - Evaluate expression into a local
    - Emit “ret” instruction

```
static int Add(int a, int b)
{
  return a + b;
}
```

```
.method static int32 Add(int32 a, int32 b)
{
  .maxstack  2
  .locals init (int32 V_0)
  IL_0000:  nop
  IL_0001:  ldarg.0
  IL_0002:  ldarg.1
  IL_0003:  add
  IL_0004:  stloc.0
  IL_0005:  br.s      IL_0007
  IL_0007:  ldloc.0
  IL_0008:  ret
}
```

Code for +

Code for return

Temporary

Sequencing

Compiler Optimizations

- **Excessive use of locals**
  - Compilation of return statement
    - Evaluate expression into a local
    - Emit “ret” instruction

```
static int Add(int a, int b)
{
  return a + b;
}
```

```
.method static int32 Add(int32 a, int32 b)
{
  .maxstack  2
  .locals init (int32 V_0)
  IL_0000:  nop
  IL_0001:  ldarg.0
  IL_0002:  ldarg.1
  IL_0003:  add
  IL_0004:  stloc.0
  IL_0005:  br.s       IL_0007
  IL_0007:  ldloc.0
  IL_0008:  ret
}
```

Can just fall through

Compiler Optimizations

- **Excessive use of locals**
  - Compilation of return statement
    - Evaluate expression into a local
    - Emit "ret" instruction

```
static int Add(int a, int b)
{
  return a + b;
}
```

```
.method static int32 Add(int32 a, int32 b)
{
  .maxstack  2
  .locals init (int32 V_0)
  IL_0000:  nop
  IL_0001:  ldarg.0
  IL_0002:  ldarg.1
  IL_0003:  add
  IL_0004:  stloc.0
  IL_0007:  ldloc.0
  IL_0008:  ret
}
```

Store to and load from same local

Compiler Optimizations

- **Excessive use of locals**
  - Compilation of return statement
    - Evaluate expression into a local
    - Emit “ret” instruction

```
static int Add(int a, int b)
{
  return a + b;
}
```

```
.method static int32 Add(int32 a, int32 b)
{
  .maxstack  2
  IL_0000:  nop
  IL_0001:  ldarg.0
  IL_0002:  ldarg.1
  IL_0003:  add
  IL_0008:  ret
}
```

Nop for { token

Compiler Optimizations

- **Excessive use of locals**
  - Compilation of return statement
    - Evaluate expression into a local
    - Emit "ret" instruction

```
static int Add(int a, int b)
{
  return a + b;
}
```

```
.method static int32 Add(int32 a, int32 b)
{
  .maxstack  2
  IL_0001:  ldarg.0
  IL_0002:  ldarg.1
  IL_0003:  add
  IL_0008:  ret
}
```

Result of /o+

Compiler Optimizations

- **Branch optimization**
  - Compilation of conditional
    - Evaluate conditional, e.g. using c* instructions, into local
    - Perform branch based on local

```
static int Div(int a, int b)
{
  if (b == 0)
    throw new Exception();

  return a / b;
}
```

Inverted condition

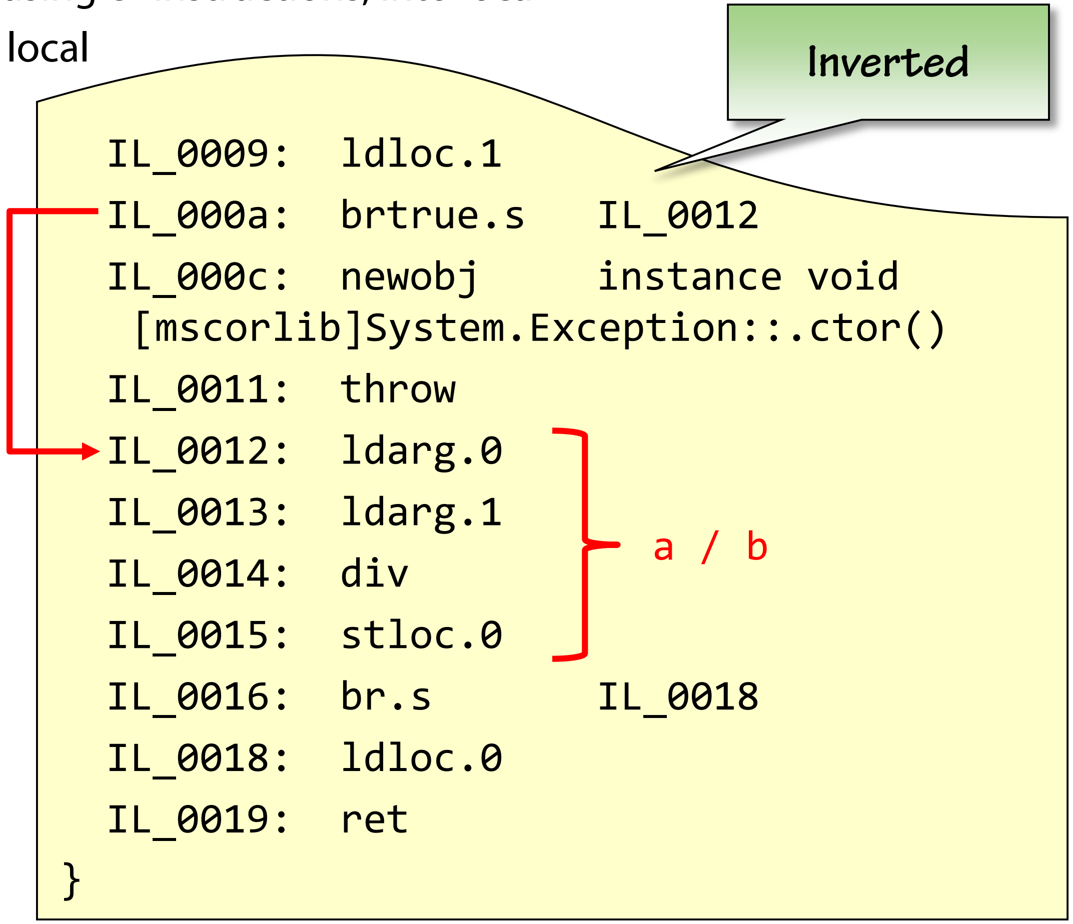
```
.method static int32 Div(int32 a, int32 b)
{
  .maxstack  8
  .locals init (int32 V_0, bool V_1)
  IL_0000:  nop
  IL_0001:  ldarg.1
  IL_0002:  ldc.i4.0       b == 0
  IL_0003:  ceq
  IL_0005:  ldc.i4.0
  IL_0006:  ceq            (b == 0) == false
  IL_0008:  stloc.1
  …
```

Compiler Optimizations

- **Branch optimization**
  - Compilation of conditional
    - Evaluate conditional, e.g. using c* instructions, into local
    - Perform branch based on local

```
static int Div(int a, int b)
{
  if (b == 0)
    throw new Exception();

  return a / b;
}
```



```
  IL_0009:  ldloc.1
  IL_000a:  brtrue.s   IL_0012
  IL_000c:  newobj     instance void
   [mscorlib]System.Exception::.ctor()
  IL_0011:  throw
  IL_0012:  ldarg.0
  IL_0013:  ldarg.1
  IL_0014:  div
  IL_0015:  stloc.0
  IL_0016:  br.s       IL_0018
  IL_0018:  ldloc.0
  IL_0019:  ret
}
```

Compiler Optimizations

- **Branch optimization**
  - Compilation of conditional
    - Evaluate conditional, e.g. using c* instructions, into local
    - Perform branch based on local

```
static int Div(int a, int b)
{
  if (b == 0)
    throw new Exception();

  return a / b;
}
```

```
.method static int32 Div(int32 a, int32 b)
{
  .maxstack  8
  IL_0000:  ldarg.1
  IL_0001:  brtrue.s   IL_0009
  IL_0003:  newobj     instance void
   [mscorlib]System.Exception::.ctor()
  IL_0008:  throw
  IL_0009:  ldarg.0
  IL_000a:  ldarg.1
  IL_000b:  div
  IL_000c:  ret
}
```

Compare to 0

Result of /o+

Compilation Targets

- **Target specified using /t switch**
  - Executable files
    - exe – console application (CUI)
      - Console Application projects
    - winexe – graphical UI application (GUI)
      - Windows Forms, WPF projects
    - appcontainerexe – WinRT application
      - Windows XAML projects
    - See dumpbin.exe for the subsystem flag used by the Windows loader
  - Library files
    - library – assembly in a .dll file
      - Class Library projects
    - module – netmodule that can be linked using al.exe
      - No Visual Studio project support
    - winmdobj – Windows Metadata (WinMD) object file
      - Windows Runtime Module projects
      - Consumed by WinMDExp.exe

Compilation Targets

- **Windows Runtime (WinRT)**
  - Evolution of COM
    - Self-describing modules using metadata
    - Classes besides interfaces
    - Application host model
    - Support for multiple languages and runtimes
      - Native code (CRT)
      - Managed code (CLR)
      - JavaScript (Chakra)
  - .winmd files
    - are <u>not</u> .NET assemblies
    - have the same metadata (ECMA 335)
    - can be ILDASM'ed
  - See %WINDIR%\System32\WinMetadata

Compilation Targets

```
C:\> ildasm.exe C:\Windows\System32\WinMetadata\Windows.System.winmd
```

```
// Metadata version: WindowsRuntime 1.3

.assembly windowsruntime Windows.System
{
  …
  .hash algorithm 0x00008004
  .ver 255:255:255:255
}

.class public abstract auto ansi windowsruntime sealed
   Windows.System.Threading.ThreadPool
        extends [mscorlib]System.Object
{
}
```

Small lie ☺

Architectures

- **Architecture specified using /platform switch**
  - C# compiler <u>always</u> generates IL code
  - Platform influences CLR header flags
  - Use corflags.exe to check flags

- **Supported architectures**
  - Default: anycpu
    - JIT or NGEN compiles to “best” target CPU architecture
    - E.g. x64 when running on AMD or Intel 64 bit
  - Specific architectures
    - x86 (will run in WOW64 on a 64-bit system)
    - x64
    - ARM, e.g. tablets running Windows RT
    - Itanium
  - New in .NET 4.5
    - anycpu32bitpreferred, will choose x86 even on a 64-bit system

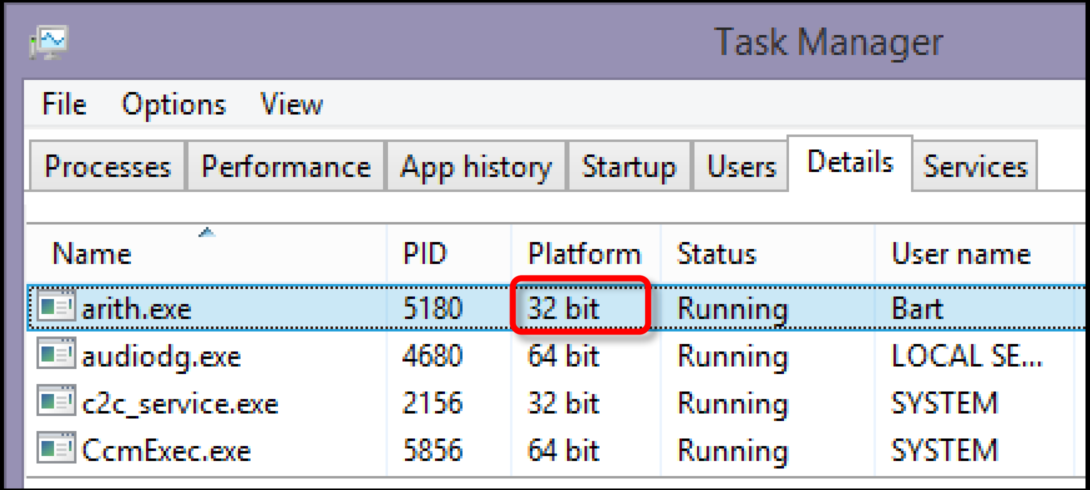
Architectures

```
C:\Demo> csc /nologo /platform:x86 arith.cs

C:\Demo> corflags /nologo arith.exe
Version    : v4.0.30319
CLR Header: 2.5
PE         : PE32
CorFlags   : 0x3
ILONLY     : 1
32BITREQ   : 1
32BITPREF  : 0
Signed     : 0

C:\Demo> arith.exe
```

/platform:x86


Task Manager
File Options View
Processes Performance App history Startup Users Details Services

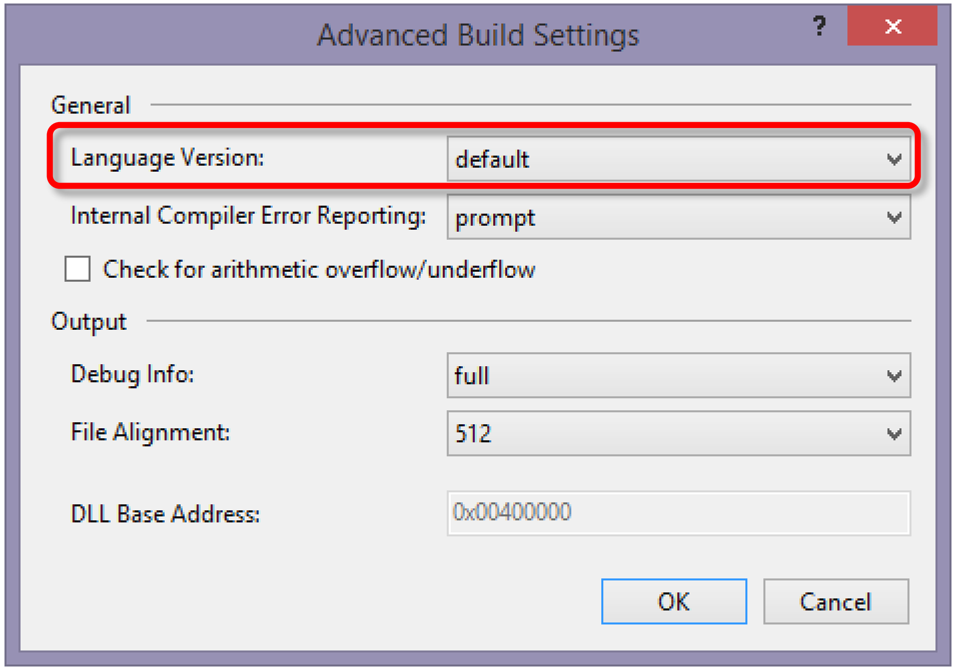
Name	PID	Platform	Status	User name
arith.exe	5180	32 bit	Running	Bart
audiodg.exe	4680	64 bit	Running	LOCAL SE...
c2c_service.exe	2156	32 bit	Running	SYSTEM
CcmExec.exe	5856	64 bit	Running	SYSTEM

Language Version Flags

- **Restrict language syntax supported using /langversion**
  - ISO-1 – C# 1.0 syntax, cf. ISO 23270:2003 specification
  - ISO-2 – C# 2.0 syntax, cf. ISO 23270:2006 specification
  - 3 – C# 3.0 syntax (LINQ, lambdas, auto properties, etc.)
  - 4 – C# 4.0 syntax (optional and named parameters, dynamic, etc.)
  - 5 – C# 5.0 syntax (async/await, caller info attributes, etc.)

- **Advanced Build Settings**
  - Not "Target Framework"!
    - Influences BCL references

- **Use to restrict language**
  - Common baseline in team
  - Optimize for <u>reading</u> code



Language Version: default

Internal Compiler Error Reporting: prompt

☐ Check for arithmetic overflow/underflow

Debug Info: full

File Alignment: 512

DLL Base Address: 0x00400000

Language Version Flags

```
C:\Demo> copy con linq.cs
using System.Linq;
class Program { static void Main() {
  var res = from x in new[] { 1, 2 } select x * x;
} }^Z

C:\Demo> csc /nologo /langversion:ISO-2 linq.cs
linq.cs(3,13): error CS1644: Feature 'query expression' cannot be
   used because it is not part of the ISO-2 C# language
   specification
linq.cs(3,23): error CS1644: Feature 'implicitly typed array'
   cannot be used because it is not part of the ISO-2 C# language
   specification

C:\Demo>
```

Assembly References

- **“Add Reference” dialog in Visual Studio**
  - Project references influence build order
    - No cycles allowed
  - /r compiler flag to reference binaries
    - Full path or file name searched on /lib paths
    - Can be used to specify aliases (cf. “extern alias”)
- **New in .NET 4.5**
  - Framework API sets and Extension SDKs
    - Referenced as a whole
    - A lot of /r flags emitted
    - Used by Portable Library, Windows Runtime SDKs, etc.
- **C# compiler prunes out what’s not used**
  - Make sure assemblies are copied if needed
  - E.g. for code relying on Assembly.LoadFrom

Extern Aliases

```
C:\Demo> notepad extern.cs
```

```
extern alias Foo1;
extern alias Foo2;

class Program
{
  static void Main()
  {
    Foo1::Bar.Qux();
    Foo2::Bar.Qux();
  }
}
```

```
// foo1.dll
public class Bar
{
  public static void Qux() {}
}
```

```
// foo2.dll
public class Bar
{
  public static void Qux() {}
}
```

```
C:\Demo> csc extern.cs /r:Foo1=foo1.dll /r:Foo2=foo2.dll
```

Assembly References

- **/nostdlib compiler switch**
  - Excludes mscorlib.dll default reference
  - Often used by Csc target in MSBuild
    - Point to specific mscorlib.dll
    - One compiler, different frameworks
- **Dependencies of C# on BCL**
  - Base classes such as Object, MulticastDelegate
  - APIs such as String.Concat, Interlocked.CompareExchange
  - Interfaces such as IDisposable
  - Etc.
- **Try creating your own mscorlib ☺**
  - …without using mscorlib, of course

Portable Library

- **Design goals**
  - “Build once, run everywhere” libraries
  - Targeting various environments
    - Desktop CLR
    - Windows XAML
    - Windows Phone
    - Silverlight
- **Refactoring of the .NET Framework**
  - Modular composition of API sets
  - Better layering and dependencies
  - Organized by functionality, not “by history”
    - E.g. System.Core
  - Hiding the assembly structure
    - Referencing just the “.NET Framework”
    - Profiles for combinations of target frameworks

Portable Library

- **Before portable library**
  - Assemblies with a bunch of stuff
  - Often not platform neutral
  - Mscorlib.dll, System.dll, System.Core.dll, etc.
  - Examples
    - HashSet<T> in System.Core.dll
    - List<T> in mscorlib.dll
- **New structure**
  - Assemblies for sets of functionality
  - Composition of assemblies into profiles
  - System.IO.dll, System.Reflection.dll, **System.Collections.dll**, etc.
- **Mechanism**
  - [assembly: TypeForwardedToAttribute]
    - Indicates a type has moved to another assembly
  - [assembly: ReferenceAssemblyAttribute]
    - Empty reference assemblies targeted by build

Summary

- **Compiler optimizations**
  - Separate pass activated by /o+
  - Branching, locals, etc.
  - There’s always the JIT!

- **Settings**
  - Compilation target using /t
  - Target platform architecture, cf. corflags.exe
  - Language version using /langversion

- **References**
  - /r to refer to required assemblies
  - Portable Library refactoring of the .NET BCL