ICT365

Software Development Frameworks

Dr Afaq Shah

Murdoch
UNIVERSITY

Common Language Infrastructure

Application Packaging and Deployment

Murdoch
UNIVERSITY

Topic Aims

Murdoch UNIVERSITY

Understand .NET assemblies

- how are they generated?
- what do they contain: manifest, metadata, IL, etc;
- the two types: executable programs (.EXE) and libraries (.DLL);
- representation of types and type references in assemblies.

Understand Common Type System (CTS)

- why common types?
- value types and reference types;
- primitive types, their names in BCL, IL and C#;
- mapping of high-level language types to CTS.

Topic Aims (cnt'd)

Murdoch UNIVERSITY

- Understand the role of Common Intermediate Language (CIL, or IL).
- Understand how high-level language types are mapped to IL types.
- Understand how CLR loads and executes .NET assemblies.
- Understand and be able to use C# compiler **`csc`** and VB compiler **`vbc`** to compile source code into .EXE and .DLL assemblies.
- Be able to use **`ildasm`** to generate textual representation of assembly code and be able to map the high-level language types to the IL types in such textual representation.

Common Language Infrastructure

Murdoch UNIVERSITY

CLR is Microsoft's implementation of Common Language Infrastructure (CLI), which defines Common Type System (CTS), metadata structure and syntax for representing CTS, and Common Intermediate Language (CIL).

CLI is an approved international standard proposed by Microsoft. The latest version is available at http://www.ecma-international.org/publications/standards/Ecma-335.htm

Framework Class Library (FCL)

Kernel Profile (defined in CLI):

- Runtime Infrastructure Library, Base Class Library (BCL)

Compact Profile (defined in CLI):

- Kernel Profile plus Network Library, XML Library, Reflection Library

Non-CLI libraries:

- ADO.NET, ASP.NET, Windows Forms

High-Level Languages for .NET

Murdoch UNIVERSITY

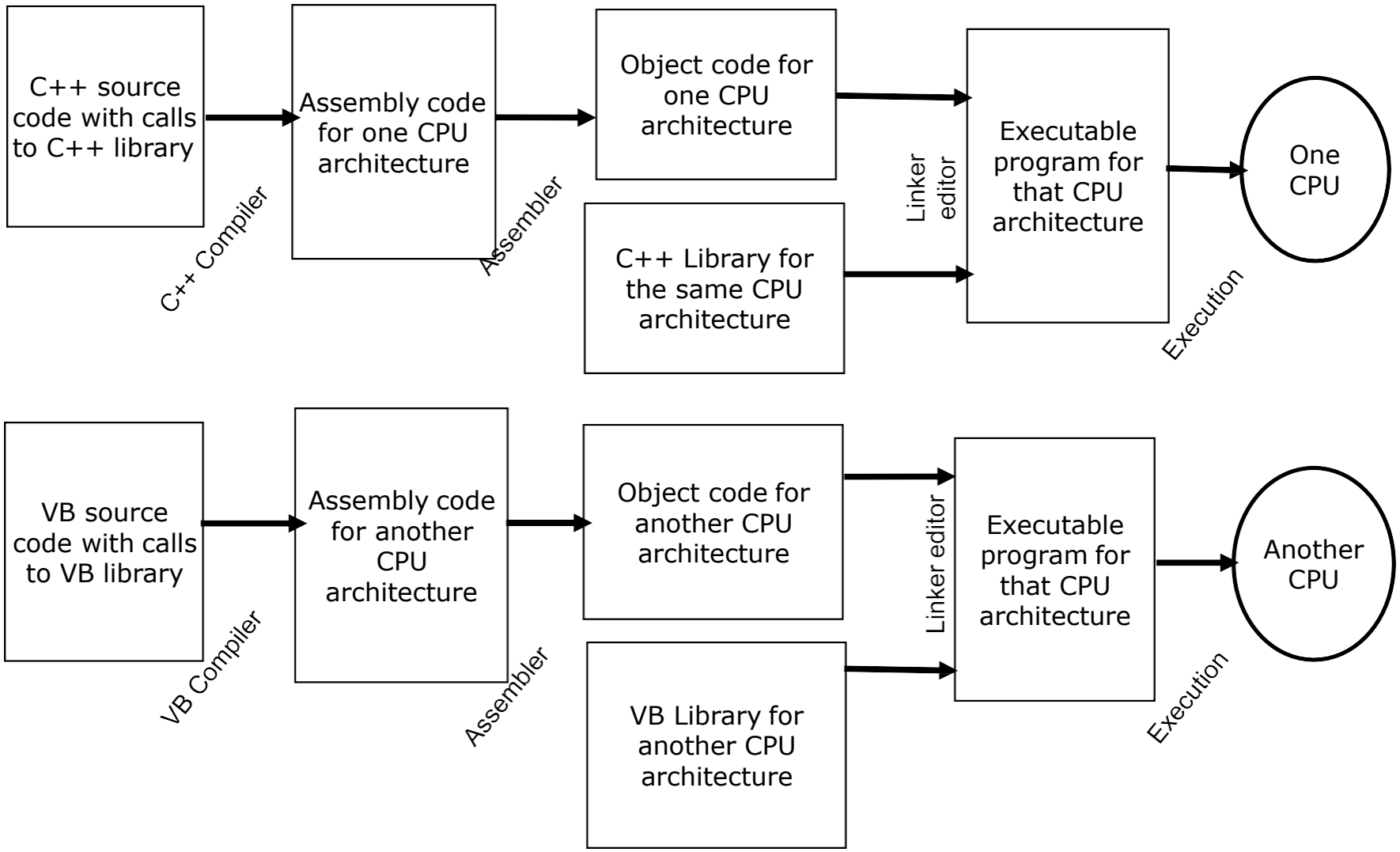
C# is the lingua franca of the .NET.

Other high-level languages are also available:

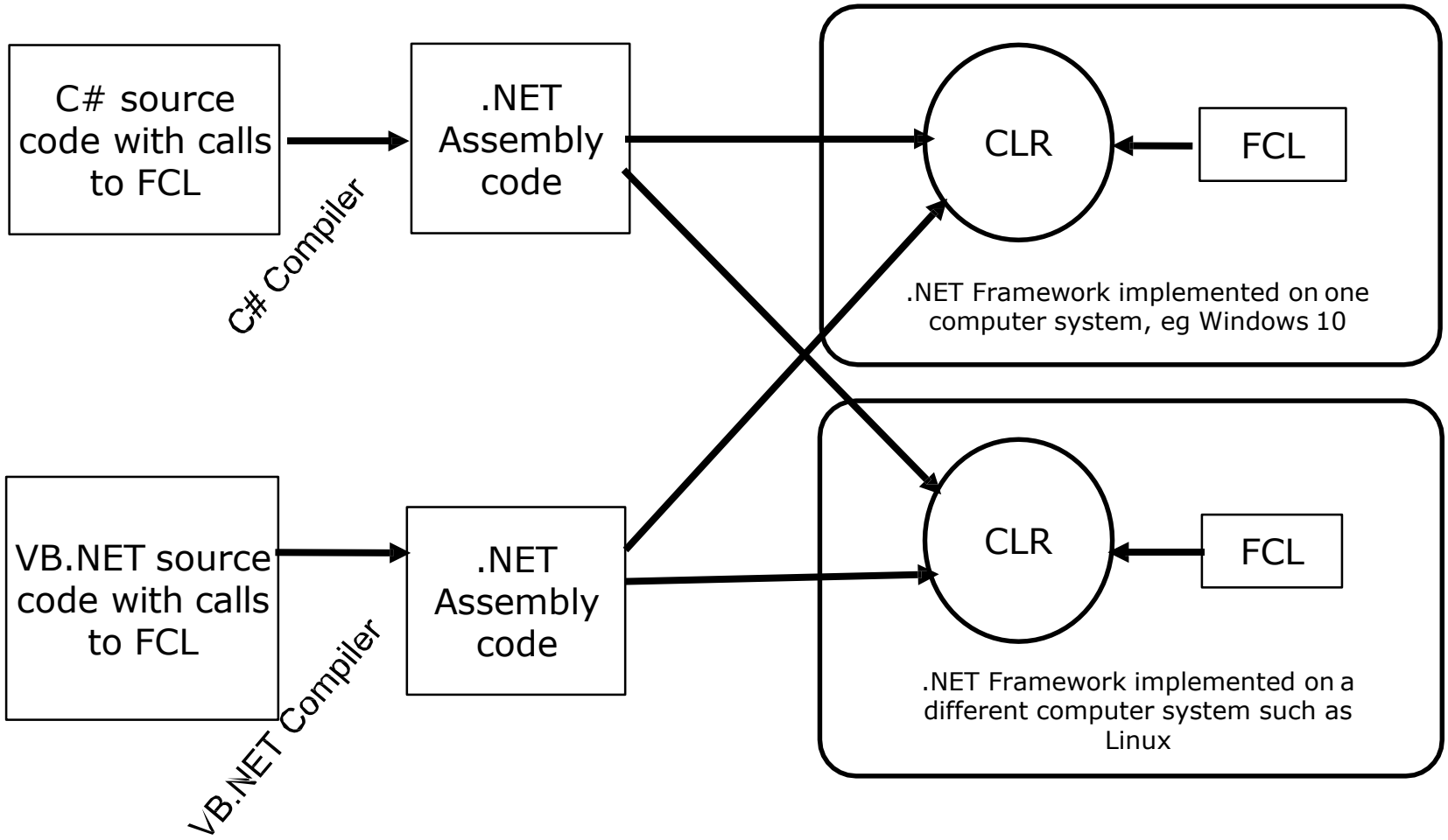
- Visual Basic .NET,
- C++/CLI,
- J#, Perl, ...

You can use the same FCL from any of the .NET languages in nearly identical way.

Traditional Programming Model



.NET Programming Model



Execution of any .NET assembly on different computer systems with possibly different operating systems and/or CPU architectures

Platform Independence

Murdoch UNIVERSITY

A .NET program is compiled into, and deployed as, an architecture independent assembly code. The format of the assembly is specified in CLI standard.

The assembly code would run on any system (Microsoft Windows, Linux, Mac OS X etc), as long as that system has an implementation of CLI and FCL.

Therefore, at least in theory, .NET programs are platform independent.

<u>In reality, complete implementation of .NET Framework is only available in Microsoft operating systems.</u>

There are attempts to implement CLI and FCL on other platforms (eg, Linux and Mac OS X) such as mono project (www.mono-project.org). However the implementation is not yet complete.

Language Interoperability

Murdoch UNIVERSITY

Under .NET, you can develop an application with one .NET language such as C#, or with several .NET languages.

You can create a new library of functionality with one .NET language (such as C#) and clients of the library can use it from any .NET language (such as VB.NET).

Once you know how to use a library from one language such as C#, you will be able to use the same library from any other .NET language.

Why Learn .NET?

Microsoft spends huge sums of money to move developers to the .NET platform.

It is expected that many new software projects will target the .NET platform, especially those that are internet and web based.

It is much quicker to develop applications on .NET platform due to its powerful class libraries.

Surveys show that there is a demand for .NET skills in the job market.

Murdoch UNIVERSITY

Simplest .NET program using C#....

```
// Hello.cs
//
// this is our first C# program

public class Hello
{
    public static void Main(string[] args)
    {
      System.Console.Out.WriteLine("Hello, world!");
    }
}
```

Compile and Run

Murdoch UNIVERSITY

Use a text editor such as Notepad++ to create the source code.

Save the source code into a file with .cs extension name, such as “Hello.cs”.

Compile it with C# compiler csc from Command Prompt:

```
csc  Hello.cs
```

Execute the program Hello.exe by typing Hello in Command Prompt or double clicking it.

Note that csc is usually under directory:

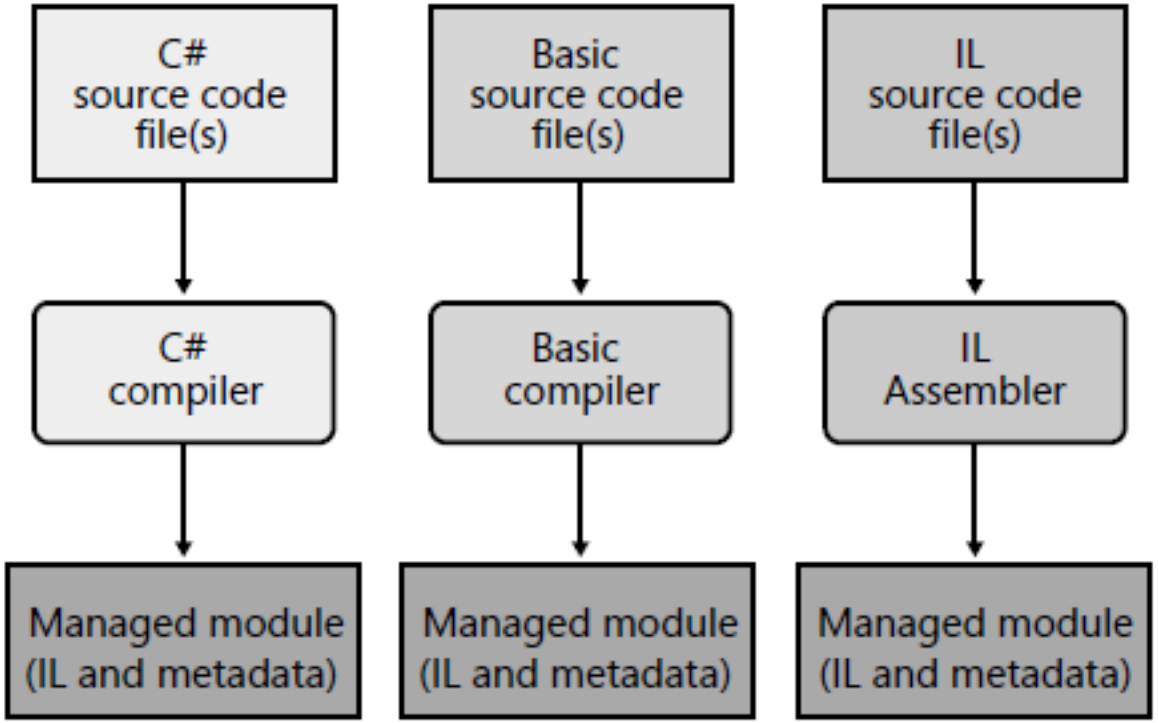
`c:\WINDOWS\microsoft.NET\Framework64\v4.0.30319\`

Example 1

Administrator: C:\Windows\System32\cmd.exe

```
c:\cs>"C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\MSBuild\15.0\Bin\Roslyn\csc
.exe"  "c:\cs\Hello.cs" /out:"c:\cs\Hello.exe"
Microsoft (R) Visual C# Compiler version 2.3.2.62116 (8522b473)
Copyright (C) Microsoft Corporation. All rights reserved.


c:\cs>dir
 Volume in drive C is OS
 Volume Serial Number is ECEE-2B07

 Directory of c:\cs

18/10/2017  12:18 PM    <DIR>          .
18/10/2017  12:18 PM    <DIR>          ..
06/10/2017  08:13 PM            47,576 csc.exe
18/10/2017  12:09 PM               189 Hello.cs
18/10/2017  12:18 PM             4,096 Hello.exe
               3 File(s)         51,861 bytes
               2 Dir(s)  270,597,541,888 bytes free

c:\cs>Hello
Hello, world!

c:\cs>
```

"C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\MSBuild\15.0\Bin\Roslyn\csc.exe" "c:\cs\Hello.cs" /out:"c:\cs\Hello.exe"



A managed module is a standard 32-bit Microsoft Windows portable executable (PE32) file or a standard 64-bit Windows portable executable (PE32+) file that requires the CLR to execute.

.NET Assembly

Murdoch UNIVERSITY

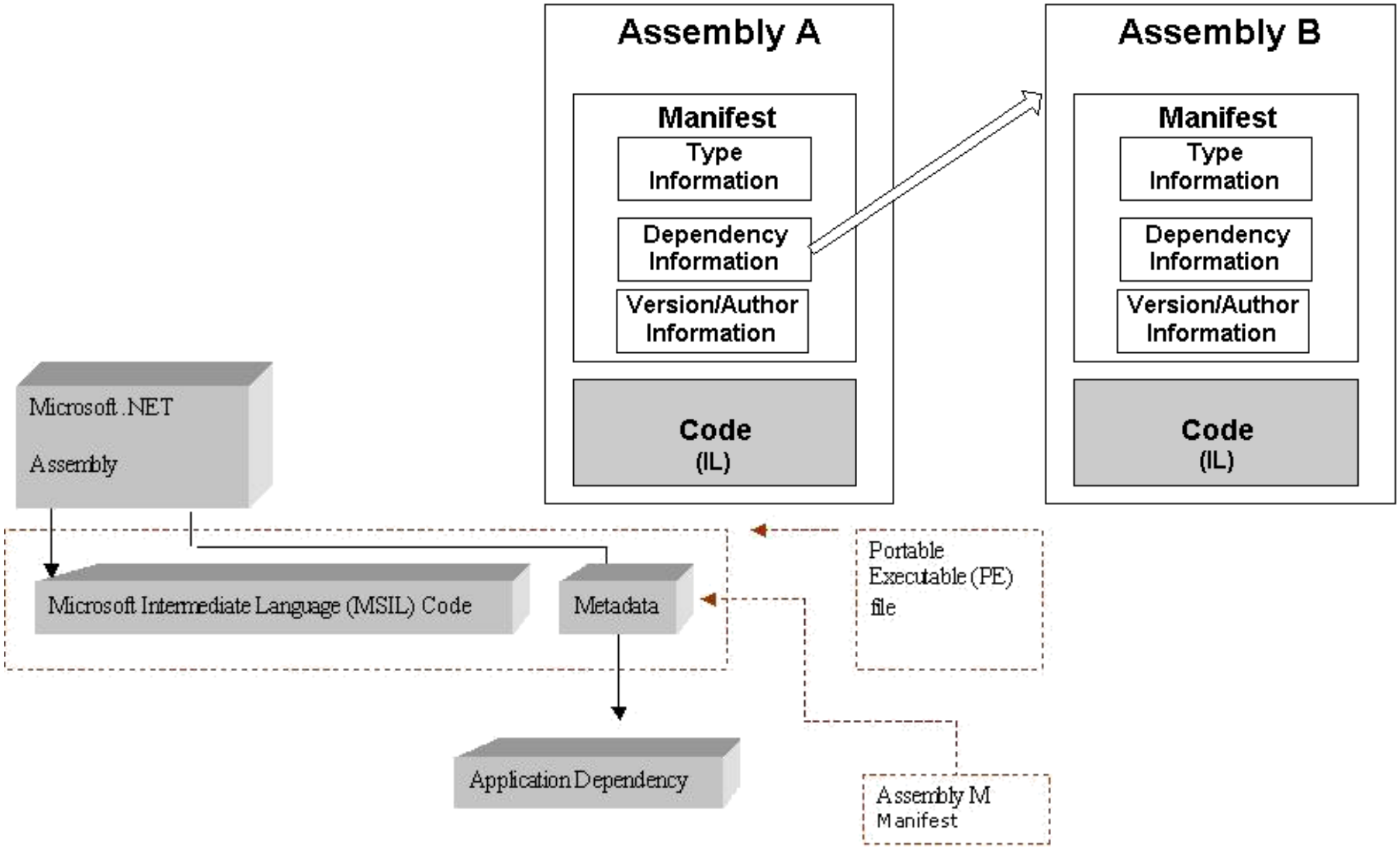
A source code written in a high-level .NET language such as C# or VB.NET, is compiled into a .NET assembly.

The assembly contains metadata representing the types that were defined and referenced in the source code as well as Common Intermediate Language (CIL, or IL) instructions that implement the methods of the types.

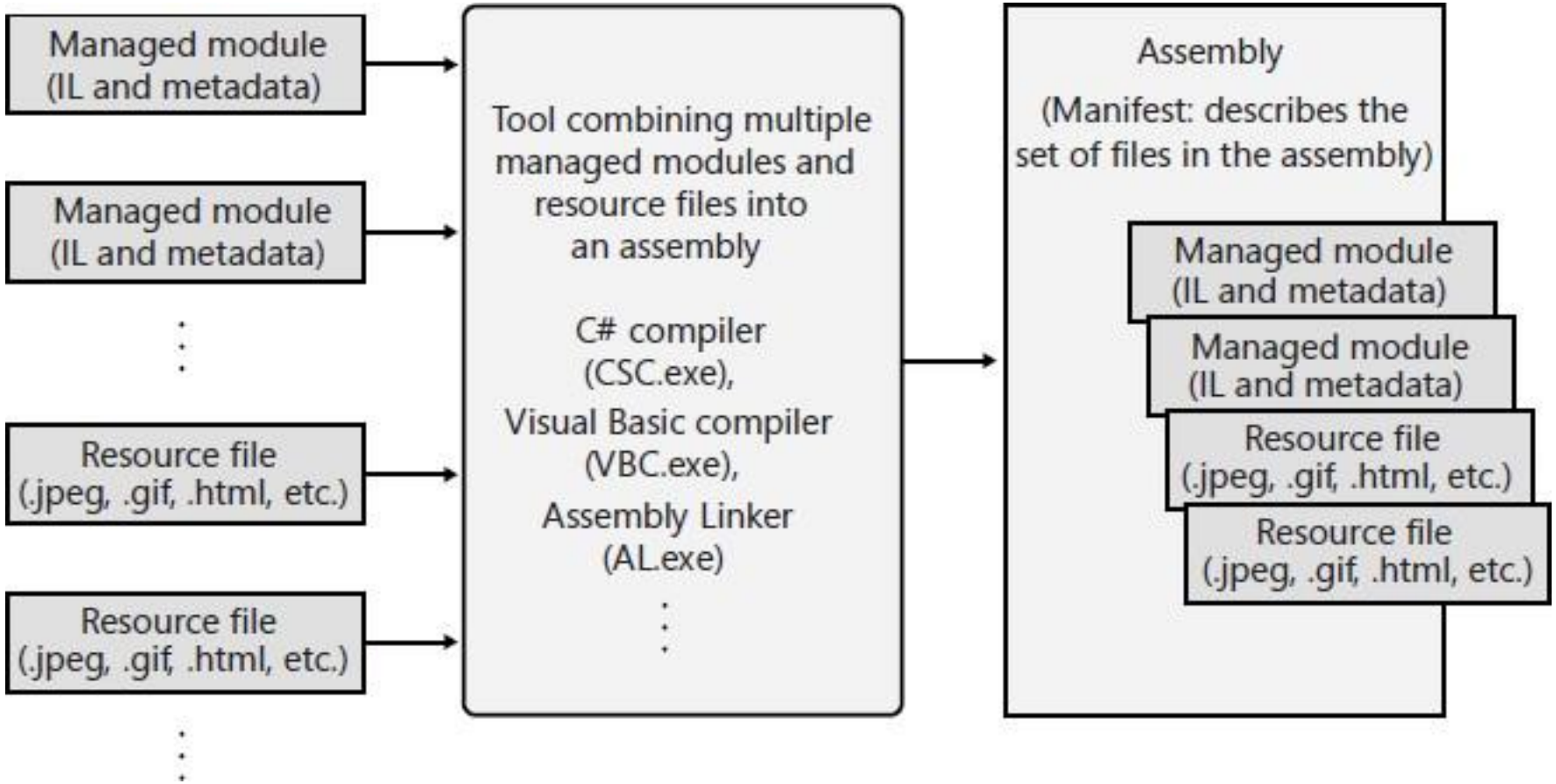
The assembly is saved into a binary file, whose file format is known as Portable Execution/Common Object File Format, or PE/COFF, or simply PE format. This is the common format for binary code on Windows.

A .NET assembly is a unit of deployment, execution and re-use.

Manifests and Assemblies



Combining managed modules into assemblies.



.EXE and .DLL Assemblies

Murdoch UNIVERSITY

There are two types of assemblies: the executable programs (.EXE) and libraries (.DLL).

.EXE assemblies differ from .DLL assemblies in that only .EXE assemblies contain a small boot-strap code to call CLR and have one and only one method with an IL directive ".entrypoint".

One can run a .EXE assembly but not .DLL assembly.

An assembly may use the types defined in other .DLL and .EXE assemblies.

Contents of Assembly

Murdoch UNIVERSITY

An assembly consists of

- a manifest: name, version etc that identifies the assembly, list of files in the assembly, list of external assemblies;
- metadata: table of type definitions and table of type references;
- IL code: methods are compiled into IL. At the runtime, the IL code is compiled into native machine code for execution;
- other types of files such as images etc.

Explore the Assembly

Murdoch UNIVERSITY

Compile the source code into an assembly:

```
csc  HelloCS.cs
```

The assembly is named HelloCS.exe. This is the default output name.

Explore the assembly using IL disassembler named ildasm.exe by typing the command:

```
ildasm  Hello.exe
```

```
c:\cs>"C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin\ildasm.exe" "c:\cs\Hello.exe"

c:\cs>
```

c:\cs\Hello.exe - IL DASM

File View Help

- c:\cs\Hello.exe
  - MANIFEST
  - Hello
    - .class public auto ansi beforefieldinit
    - .ctor : void()
    - Main : void(string[])

```
.assembly Hello
{
  .ver 0:0:0:0
```

Hello::.ctor : void()

Find Find Next

```
.method public hidebysig specialname rtspecialname
        instance void  .ctor() cil managed
{
  // Code size       8 (0x8)
  .maxstack  8
  IL_0000:  ldarg.0
  IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
  IL_0006:  nop
  IL_0007:  ret
} // end of method Hello::.ctor
```

Note: the above screenshot was generated with Visual Studio 2017 If you use other version of Visual Studio, the display diagram and its content may differ slightly.

Murdoch

MANIFEST

Find Find Next

```
// Metadata version: v2.0.50727
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                          // .z\V.
  .ver 2:0:0:0
}
.assembly HelloCS
{
  .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationRe
  .custom instance void [mscorlib]System.Runtime.CompilerServices.RuntimeCompat

  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module HelloCS.exe
// MVID: {D8B2AE28-473F-4E1C-BADA-A8AB3C091FC9}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003       // WINDOWS_CUI
.corflags 0x00000001    //  ILONLY
// Image base: 0x02EF0000
```

this is external assembly C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\mscorlib.dll

This is our assembly name

Our assembly is stored in file: HelloCS.exe

metadata .class directive for Hello class. This class is inherited from System.Object class from mscorlib assembly

Hello::.class public auto ansi beforefieldinit

Find Find Next

```
.class public auto ansi beforefieldinit Hello
        extends [mscorlib]System.Object
{
} // end of class Hello
```

metadata .method directive for the default constructor method in Hello class.
.ctor() indicates that this method is a constructor.

Hello::method .ctor : void()

Find Find Next

```
.method public hidebysig specialname rtspecialname
        instance void  .ctor() cil managed
{
  // Code size       7 (0x7)
  .maxstack  8
  IL_0000:  ldarg.0
  IL_0001:  call     instance void [mscorlib]System.Object::.ctor()
  IL_0006:  ret
} // end of method Hello::.ctor
```

This following .method directive describes another method: Main. This method has a .entrypoint directive, so the execution of the program starts from here.

Hello::method Main : void()

Find Find Next

```
.method public hidebysig static void  Main() cil managed
{
  .entrypoint
  // Code size       13 (0xd)
  .maxstack  8
  IL_0000:  nop
  IL_0001:  ldstr      "Hello, world!"
  IL_0006:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_000b:  nop
  IL_000c:  ret
} // end of method Hello::Main
```

These are IL code implementing the two methods

Note

Murdoch UNIVERSITY

You will notice *minor differences* between the examples using Visual C# 2013, 2015, 2017.

"C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\MSBuild\15.0\Bin\Roslyn\csc.exe" "c:\cs\Hello.cs" /out:"c:\cs\Hello.exe"

"C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin\ildasm.exe" "c:\cs\Hello.exe"

Example 2

Murdoch UNIVERSITY

A Visual Basic .NET Source code: HelloVB.vb

```
Option Strict On
Option Explicit On

Module Hello

    Sub Main()

        System.Console.WriteLine("hello, world!")

    End Sub

End Module
```

Note t e same metho d from the Main m ethod in the C # code

```
N:\www\lectures\t02>dir HelloVB*
 Volume in drive N is Shared Folders
 Volume Serial Number is 0000-0064

 Directory of N:\www\lectures\t02

10/02/2008  04:37 PM               156 HelloVB.vb
               1 File(s)            156 bytes
               0 Dir(s)  44,556,414,976 bytes free

N:\www\lectures\t02>vbc HelloVB.vb
Microsoft (R) Visual Basic Compiler version 9.0.21022.8
Copyright (c) Microsoft Corporation.  All rights reserved.


N:\www\lectures\t02>dir HelloVB*
 Volume in drive N is Shared Folders
 Volume Serial Number is 0000-0064

 Directory of N:\www\lectures\t02

10/02/2008  04:37 PM             6,656 HelloVB.exe
10/02/2008  04:37 PM               156 HelloVB.vb
               2 File(s)          6,812 bytes
               0 Dir(s)  44,556,410,880 bytes free

N:\www\lectures\t02>HelloVB
hello, world!

N:\www\lectures\t02>
```

c:\cs\HelloVB.exe - IL DASM

File View Help

```
c:\cs\HelloVB.exe
    MANIFEST
    My
        My.MyApplication
            .class private auto ansi
            extends [Microsoft.VisualBasic]Microsoft.VisualBasic.ApplicationServices.ApplicationBase
            .custom instance void [System]System.CodeDom.Compiler.GeneratedCodeAttribute::.ctor(string, ...
            .custom instance void [System]System.ComponentModel.EditorBrowsableAttribute::.ctor(valuetype [System]System.ComponentModel.EditorBrowsableState) = ( 01 00 01 00 00 00 00 00 ) ...
            .ctor : void()
        My.MyComputer
        My.MyProject
            .class private auto ansi sealed beforefieldinit
            .custom instance void [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.StandardModuleAttribute::.ctor() = ( 01 00 00 00 ) ...
            .custom instance void [Microsoft.VisualBasic]Microsoft.VisualBasic.HideModuleNameAttribute::.ctor() = ( 01 00 00 00 ) ...
            .custom instance void [System]System.CodeDom.Compiler.GeneratedCodeAttribute::.ctor(string, ...
            MyWebServices
            ThreadSafeObjectProvider`1<.ctor T>
            m_AppObjectProvider : private static initonly class My.MyProject/ThreadSafeObjectProvider`1<class My.MyApplication>
            m_ComputerObjectProvider : private static initonly class My.MyProject/ThreadSafeObjectProvider`1<class My.MyComputer>
            m_MyWebServicesObjectProvider : private static initonly class My.MyProject/ThreadSafeObjectProvider`1<class My.MyProject/MyWebServices>
            m_UserObjectProvider : private static initonly class My.MyProject/ThreadSafeObjectProvider`1<class [Microsoft.VisualBasic]Microsoft.VisualBasic.ApplicationServices.User>
            .cctor : void()
            get_Application : class My.MyApplication()
            get_Computer : class My.MyComputer()
            get_User : class [Microsoft.VisualBasic]Microsoft.VisualBasic.ApplicationServices.User()
            get_WebServices : class My.MyProject/MyWebServices()
            Application : class My.MyApplication()
            Computer : class My.MyComputer()
            User : class [Microsoft.VisualBasic]Microsoft.VisualBasic.ApplicationServices.User()
            WebServices : class My.MyProject/MyWebServices()
    Hello
        .class private auto ansi sealed
        .custom instance void [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.StandardModuleAttribute::.ctor() = ( 01 00 00 00 ) ...
        Main : void()
```

```
.assembly HelloVB
{
  .ver 0:0:0:0
}
```

Command Line Options

Murdoch UNIVERSITY

The compilers csc and vbc have many command line options. Use csc /help and vbc /help to find out those options.

For example, you can create a library assembly with option /target:library.

The disassembler ildasm also has many options. For example /out allows you to create textual file, rather than using GUI.

Example 3

Murdoch UNIVERSITY

A C# code for a library class MyClass.cs.

This source code can be compiled into a library assembly.

The following command compiles the source code into a library assembly named MyClassLib.dll:

```
// MyClass.cs
// .....

public class MyClass
{
    public static string Answer()
    {
        return "My name is…";
    }
}
```

```
csc  /target:library  /out:MyClassLib.dll  MyClass.cs
```

Example 4

A C# Source code: Question.cs

```
// Question.cs
//
// calling a method in MyClass

public class Hello
{
    public static void Main()
    {
      System.Console.WriteLine("Hello, what's your name?");
      string msg = MyClass.Answer();
      System.Console.WriteLine("The answer is {0}.", msg);
    }
}
```

Note that we are calling method Answer which is defined in class MyClass inside a separate library assembly named MyClassLib.dll

Copy (3) of Command Prompt

```
The answer is My name is Hong.

C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>
C:\cygwin\home\Administrator\ict239\MyClass>csc /reference:MyClassLib.dll Questi
on.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.


C:\cygwin\home\Administrator\ict239\MyClass>Question
Hello, what's your name?
The answer is My name is Hong.

C:\cygwin\home\Administrator\ict239\MyClass>_
```

Use /Reference Option

Murdoch UNIVERSITY

Since Question.cs uses the class MyClass from library assembly MyClassLib.dll, it is necessary to specify the location of that library assembly in order to compile Question.cs successfully:

```
csc /reference:MyClassLib.dll  Question.cs
```

As a general rule, you should specify all external assemblies when compiling your source code except mscorlib.dll which is included automatically by the compiler.

Compiling Multiple Files into One Assembly

We can also compile several source code files into one assembly.

For instance, we can compile both Question.cs and MyClass.cs into one assembly Question.exe:

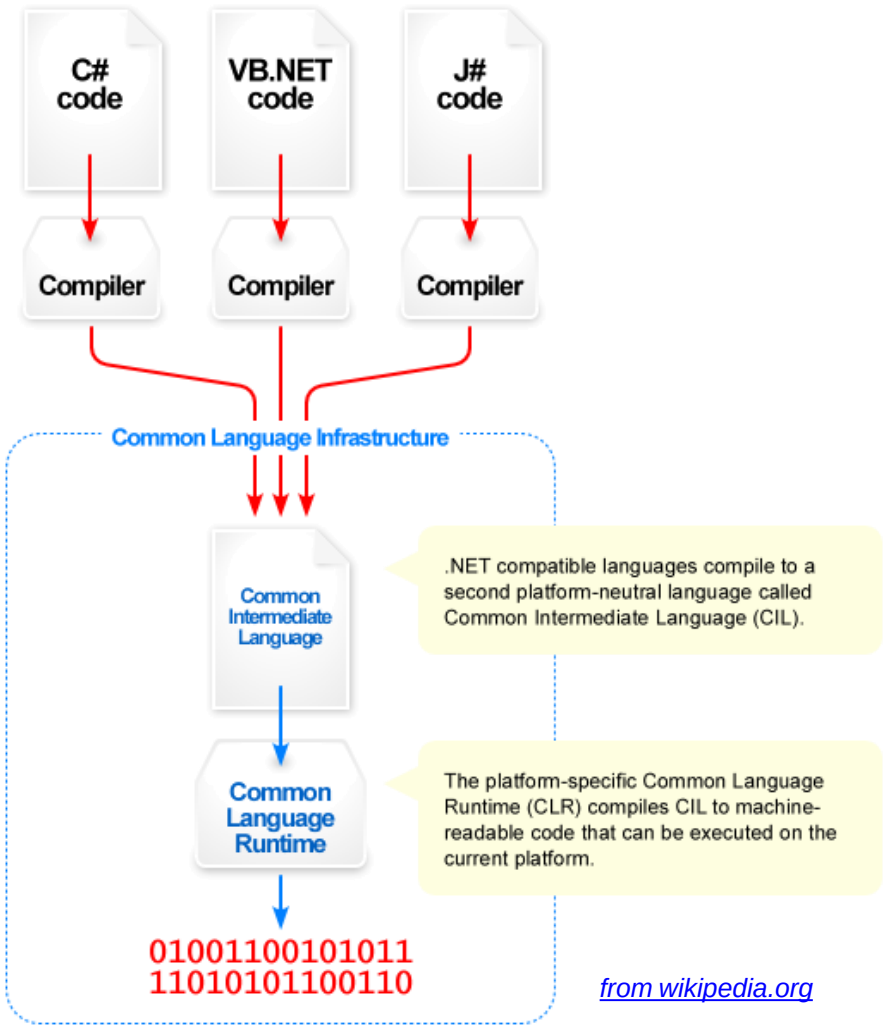
```
csc Question.cs MyClass.cs
```

CLR versus CLI.

CLR is actually an **implementation** by Microsoft of the CLI (Common Language Infrastructure) .

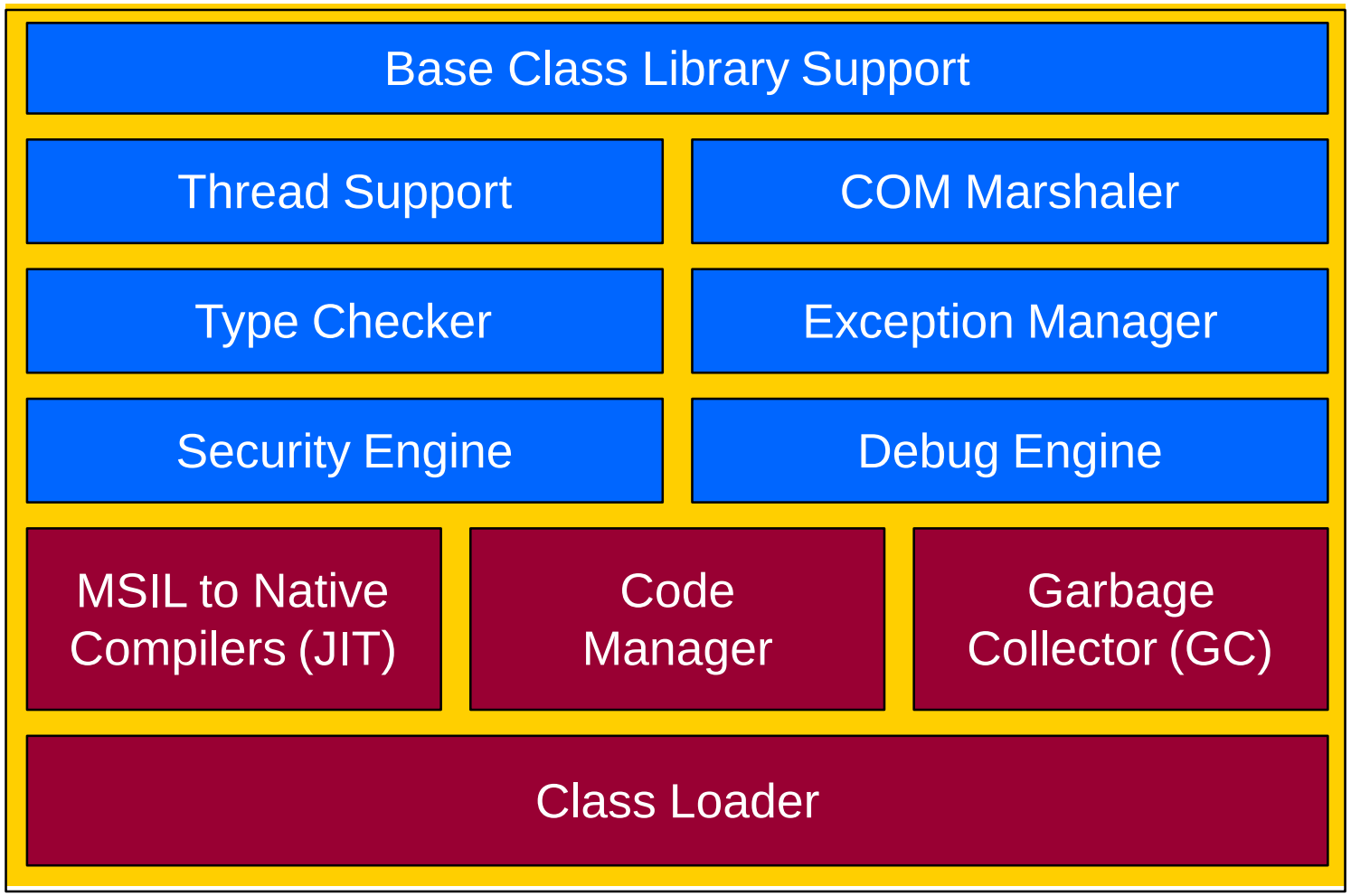
CLI is an open *specification*.

CLR is really a platform specific implementation.



from wikipedia.org

The CLR Architecture



From MSDN

Common Language Infrastructure (CLI)

Murdoch UNIVERSITY

- CLI allows for cross-language development.
- Four components:

Common Type System (CTS)

Meta-data in a language agnostic fashion.

Common Language Specification – behaviors that all languages need to follow.

A Virtual Execution System (VES).

Common Type System (CTS)

Murdoch UNIVERSITY

- A specification for *how* types are *defined* and how they *behave*.

no syntax specified

- A type can contain zero or more members:

Field

Method

Property

Event

- We will go over these more throughout the quarter.

Common Type System (CTS)

- CTS also specifies the rules for visibility and access to members of a type:

Private

Family

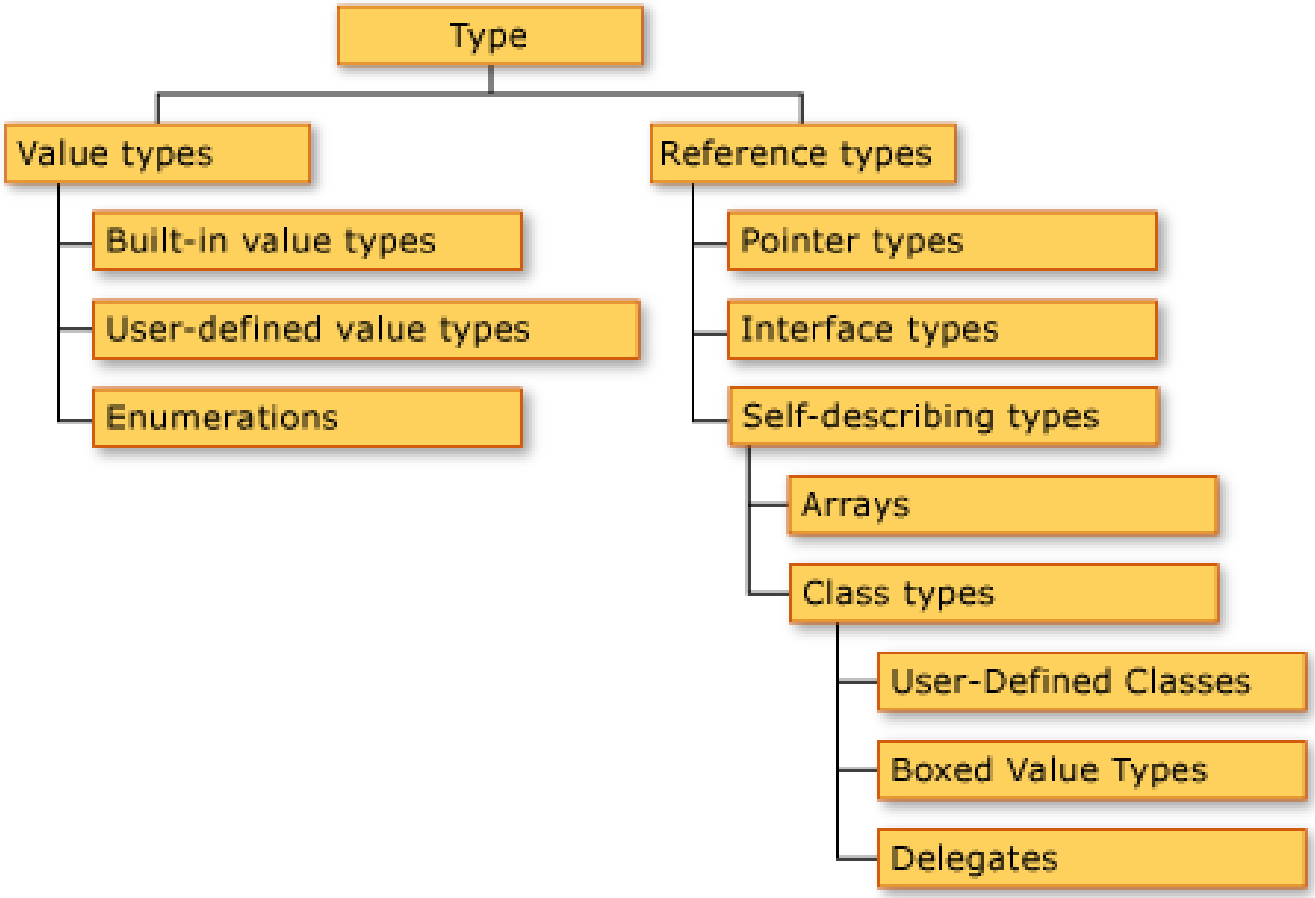
Family and Assembly

Assembly

Family or Assembly

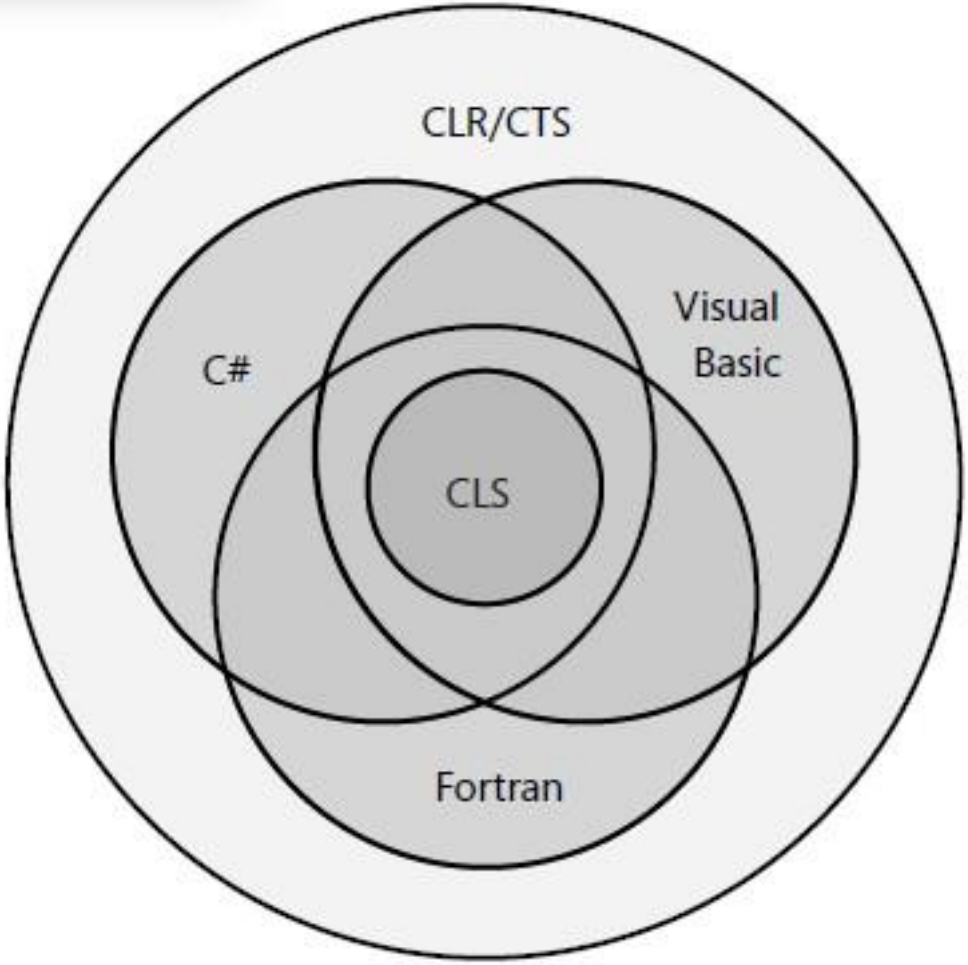
Public

Common Type System (CTS)



From MSDN

Languages offer a subset of the CLR/CTS and a superset of the CLS (but not necessarily the same superset).



The CLR/CTS supports a lot more features than the subset defined by the CLS, so if you don't care about interlanguage operability, you can develop very rich types limited only by the language's feature set. Specifically, the CLS defines rules that externally visible types and methods must adhere to if they are to be accessible from any CLS-compliant programming language. Note that the CLS rules don't apply to code that is accessible only within the defining assembly.

Common Type System

Murdoch UNIVERSITY

One of the primary aims of .NET is language interoperability.

A major obstacle in language interoperability is the existence of many similar, but incompatible, types in different high-level programming languages.

To achieve language interoperability, the underlying CLR must support a common set of types into which the types from all high-level languages can be mapped.

CLI specifies just such a common set of types known as Common Type System or CTS.

Common Type System

Murdoch UNIVERSITY

Another important reason that calls for CTS is the need to creating a common set of libraries that can be used from, and created by, any .NET language.

Without a common type system it would be impossible to create such a set of libraries.

The best way to organise such kind of libraries is the object oriented system, due to its excellent support for encapsulation, inheritance, and polymorphism.

Therefore CTS must be an object oriented system.

Taxonomy of CTS

CTS consists of value types and reference types.

Value types are referenced directly in the program stack.

Reference types are stored in program heap and are referenced via a pointer.

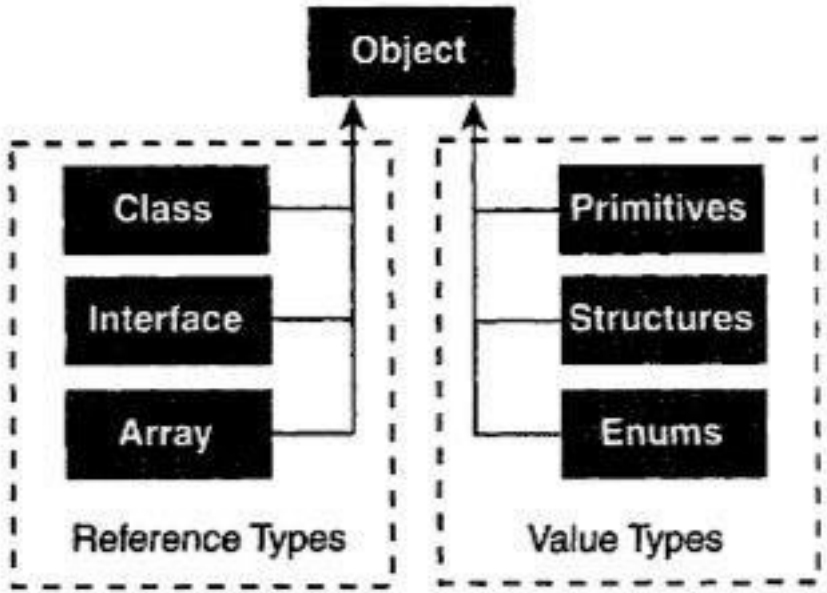


Figure 1-4 Base types defined by Common Type System

The above diagram is copied from Unit Reader: Intro to .NET and C#

Built-in Types

Murdoch UNIVERSITY

BCL Type	C#	VB	C++	IL
Boolean	bool	Boolean	bool	bool
Byte	byte	Byte	unsigned char	unsigned int8
Char	char	Char	wchar_t	char
DateTime	n/a	Date	n/a	n/a
Decimal	decimal	Decimal	n/a	n/a
Double	double	Double	double	float64
Int16	short	Short	short	int16
Int32	int	Integer	int	int32

Built-in Types

Murdoch UNIVERSITY

BCL Type	C#	VB	C++	IL
Int64	long	Long	int64	int64
IntPtr	n/a	n/a	n/a	native int
Object	object	Object	n/a	object
SByte	sbyte	n/a	signed char	int8
Single	float	Single	float	float32
String	string	String	n/a	string
UInt16	ushort	n/a	unsigned short	unsigned int16
UInt32	uint	n/a	unsigned int	unsigned int32

Built-in Types

BCL Type	C#	VB	C++	IL
UInt64	ulong	n/a	unsigned int64	unsigned int64
UIntPtr	n/a	n/a	n/a	native unsigned int

Note:

1) all primitive types lives in System namespace in mscorlib.dll assembly

2) all types listed in the above table except Object and String are value types

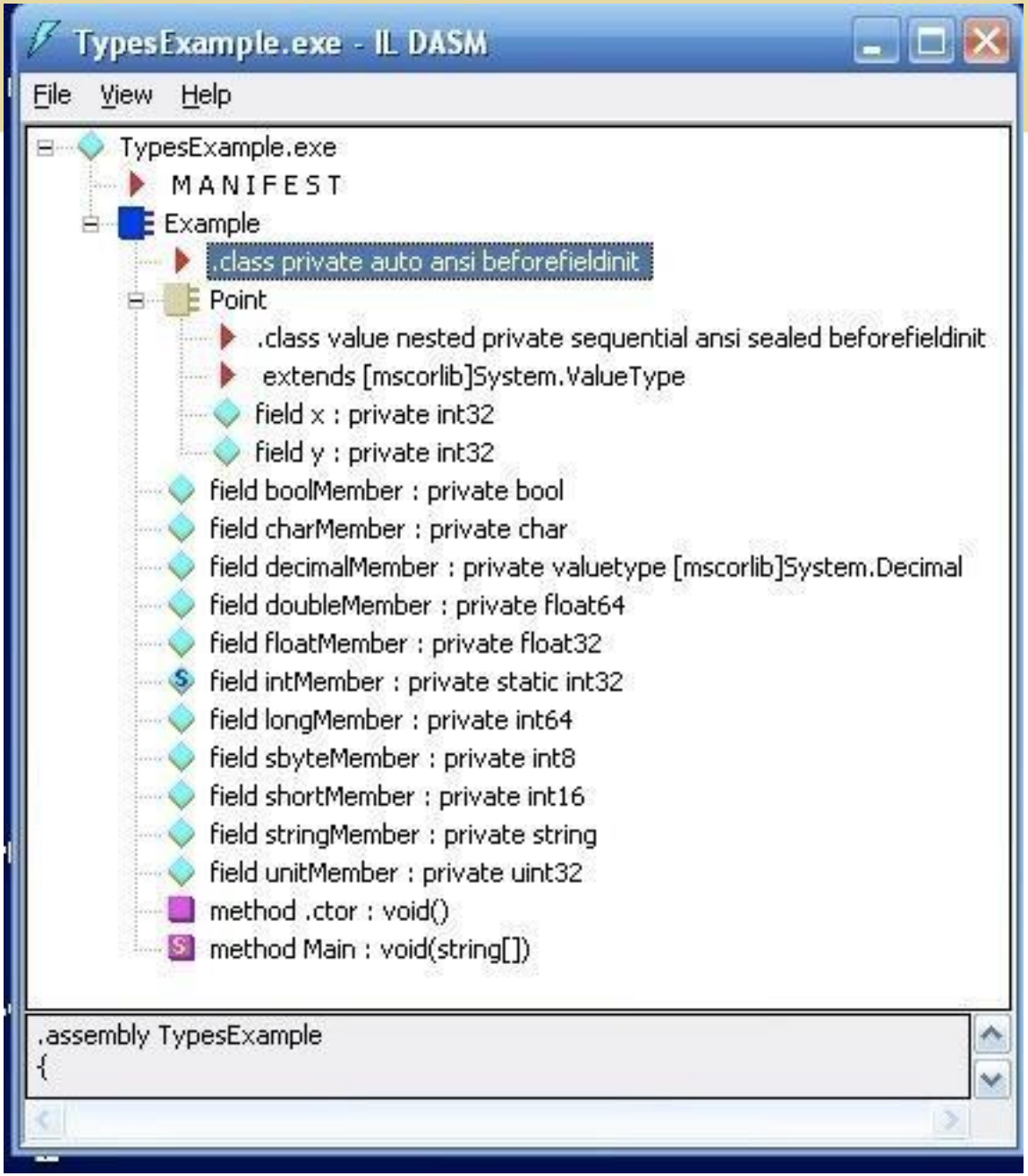
Example 5

The following example defines many variables of different primitive types in C#

```
// TypesExample.cs
//
// compare how types in C# are matched
// to IL

using System;

class Example
{
    static int intMember;
    long longMember;
    uint unitMember;
```

```
    float floatMember;
    double doubleMember;
    char charMember;
    bool boolMember;
    short shortMember;
    decimal decimalMember;
    sbyte sbyteMember;
    string stringMember;

    struct Point
    {
        int x;
        int y;
    }
```

```
  static void Main(string[] args)
    {
        int intLocal;
        long longLocal;
        uint uintLocal;
        float floatLocal;
        double doubleLocal;
        char charLocal;

            Console.WriteLine("Beginning of Example");
        intMember = 32;
        intLocal = -10;
        longLocal = 20;
        uintLocal = 100;
        floatLocal = 15.5F;
        doubleLocal = 31.4;
        charLocal = 'A';
        Console.WriteLine("intLocal = {0}", intLocal);
        Console.WriteLine("intMember = {0}", intMember);
        Console.WriteLine("The end of Example");
    }
}
```


TypesExample.exe - IL DASM
File View Help
TypesExample.exe
MANIFEST
Example
.class private auto ansi beforefieldinit
Point
.class value nested private sequential ansi sealed beforefieldinit
extends [mscorlib]System.ValueType
field x : private int32
field y : private int32
field boolMember : private bool
field charMember : private char
field decimalMember : private valuetype [mscorlib]System.Decimal
field doubleMember : private float64
field floatMember : private float32
field intMember : private static int32
field longMember : private int64
field sbyteMember : private int8
field shortMember : private int16
field stringMember : private string
field unitMember : private uint32
method .ctor : void()
method Main : void(string[])
.assembly TypesExample
{

MANIFEST

Find Find Next

```
// Metadata version: v2.0.50727
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )                         // .z\V
  .ver 2:0:0:0
}
.assembly TypesExample
{
  .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationR
  .custom instance void [mscorlib]System.Runtime.CompilerServices.RuntimeCompa

  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module TypesExample.exe
// MVID: {F12213CB-BC4E-44CB-ABCE-953480B4CE42}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003       // WINDOWS_CUI
.corflags 0x00000001    //  ILONLY
// Image base: 0x02EF0000
```

Example::method Main : void(string[])

Find Find Next

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       100 (0x64)
  .maxstack  2
  .locals init (int32 V_0,
           int64 V_1,
           uint32 V_2,
           float32 V_3,
           float64 V_4,
           char V_5)
  IL_0000:  nop
  IL_0001:  ldstr      "Beginning of Example"
  IL_0006:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_000b:  nop
  IL_000c:  ldc.i4.s   32
  IL_000e:  stsfld     int32 Example::intMember
  IL_0013:  ldc.i4.s   -10
  IL_0015:  stloc.0
  IL_0016:  ldc.i4.s   20
  IL_0018:  conv.i8
  IL_0019:  stloc.1
  IL_001a:  ldc.i4.s   100
  IL_001c:  stloc.2
  IL_001d:  ldc.r4     15.5
  IL_0022:  stloc.3
  IL_0023:  ldc.r8     31.399999999999999
  IL_002c:  stloc.s    V_4
  IL_002e:  ldc.i4.s   65
  IL_0030:  stloc.s    V_5
  IL_0032:  ldstr      "intLocal = {0}"
  IL_0037:  ldloc.0
  IL_0038:  box        [mscorlib]System.Int32
  IL_003d:  call       void [mscorlib]System.Console::WriteLine(string,
                                                                object)
  IL_0042:  nop
```

Common Intermediate Language

Murdoch UNIVERSITY

- Common Intermediate Language (CIL or IL) is specified in CLI and is implemented in CLR.
- It is similar to many assembly languages but it is not targeting any specific processor. This makes .NET programs processor-independent.
- The IL operates as a stack machine in that most operands are pushed into the stack, and instructions make use of these operands from stack rather than from registers. The later is how most CPU architectures operate with.
- The stack machine makes IL more general purpose as one does not need to worry about how many registers should be available in the underlying hardware.
- The code based on stack machine can be efficiently compiled to register-based CPU.

Just-In-Time Compilation

Murdoch UNIVERSITY

The IL code cannot be executed directly on any processor. It is compiled into native CPU instructions at run-time when the method is called.

Such compilation is known as Just-In-Time compilation. The compiler is commonly referred to as JIT. JIT is a component of CLR.

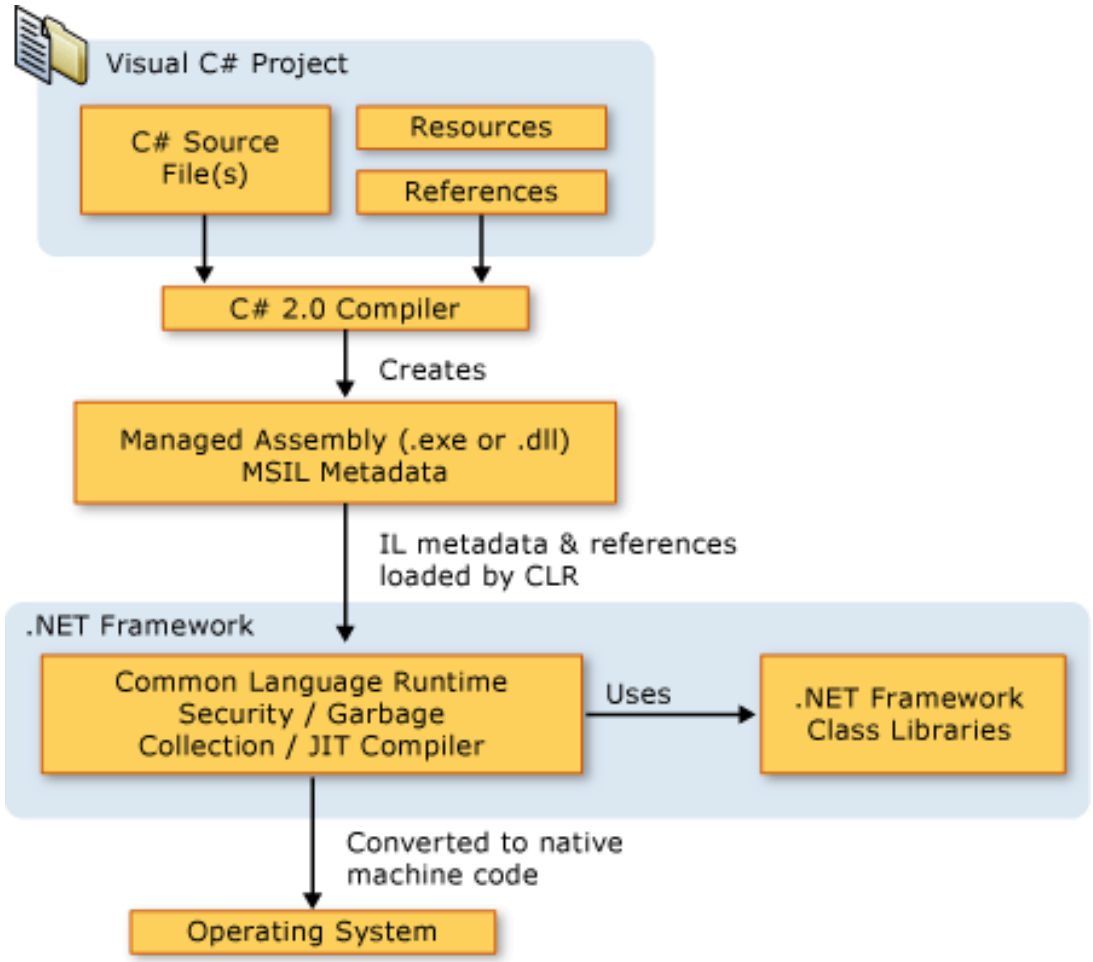
A method is only compiled once in a process when it is called the first time.

Subsequent calls to the same method will directly call the compiled native code, rather than the IL code.

CLR and JIT compiling.

Murdoch UNIVERSITY

- C#, like Java, is executed indirectly through an abstract computer architecture called the CLR.
  - CLR => Common Language Runtime.
  - Abstract, but well defined.
- C# programs are compiled to an IL.
  - Also called MSIL, CIL (Common Intermediate Language) or bytecode.



http://msdn2.microsoft.com/en-us/library/z1zx9t92(VS.80).aspx

When does CLR start?

Murdoch UNIVERSITY

The .EXE assembly contains a tiny boot-strap code.

When the assembly is loaded, that boot-strap code is executed. This code loads CLR and pass the Main method (or the method with an .entrypoint directive) to it.

From there CLR would take control of the execution of the assembly.

Therefore CLR is transparent to the user.

Application Packaging and Deployment

Application Deployment

Murdoch UNIVERSITY

Need to consider the following issues:

- how the application is packaged
- how the packaged application is distributed:
  - on a portable media such as a CD, or a DVD, or a USB drive
  - downloadable from network such as website or network share
  - retrievable from network selectively by the installation program
- how the application is run on the local machine:
  - by installing on the local machine
  - by running directly off the network such as a website
- how the application is updated

ClickOnce Technology

Murdoch UNIVERSITY

- .NET Framework 2 introduced ClickOnce deployment technology
- Visual Studio can package an application for distribution
- A packaged application can be distributed using a portable media such as DVD, a network share, or a website
- An application can be installed on a local machine under the user's profile, so that it can be run by the user off-line
- An application can also be launched from a website, similar to Java Webstart
- An application can automatically check for the new version on the website and update itself

Deployment Using Portable Media

Murdoch UNIVERSITY

This example demonstrates how to deploy an application using a CD or similar portable media.

After the application is packaged (i.e., published) in a disk directory, you can copy the directory files to a CD or a USB drive for distribution.

To install this application, run “setup.exe” program included in the package.

The application will be installed on the logon user’s profile on the local machine in an obfuscated location. A shortcut will be added to the start menu. An entry will be added to the Add/Remove Program in Windows Control.

The application can be removed using Add/Remove Program.

Create Deployment Package

Murdoch UNIVERSITY

From Visual Studio's Solution Explorer window, right-click the project, then select "Publish"

"Publish Wizard" dialog pops up. Select the directory to temporarily store the application package. Then click Next.

Click "From CD-ROM or DVD-ROM" radio button, then click Next

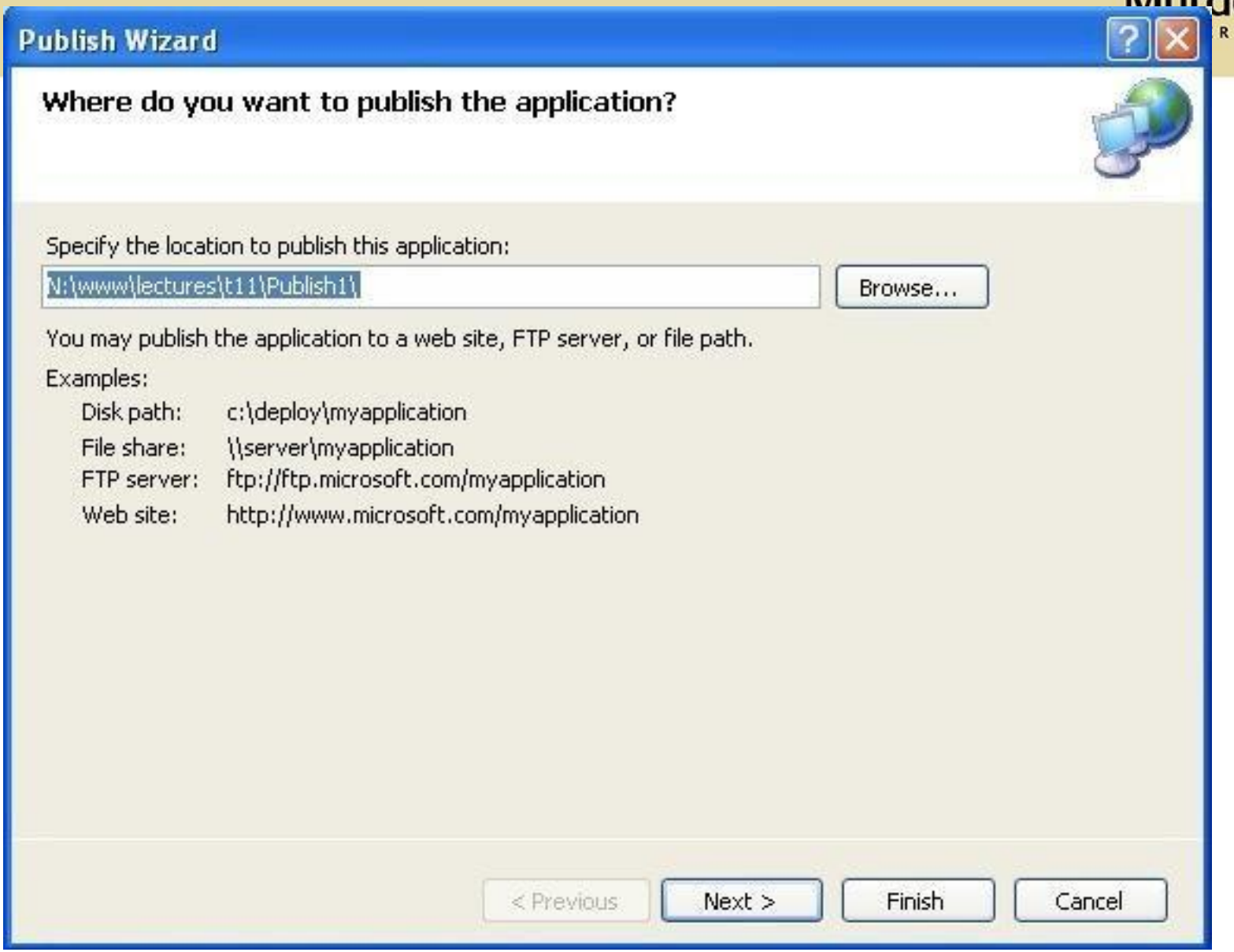
Click "The application will not check for updates", then click Next

The dialog shows "Ready to Publish!". Click Finish.

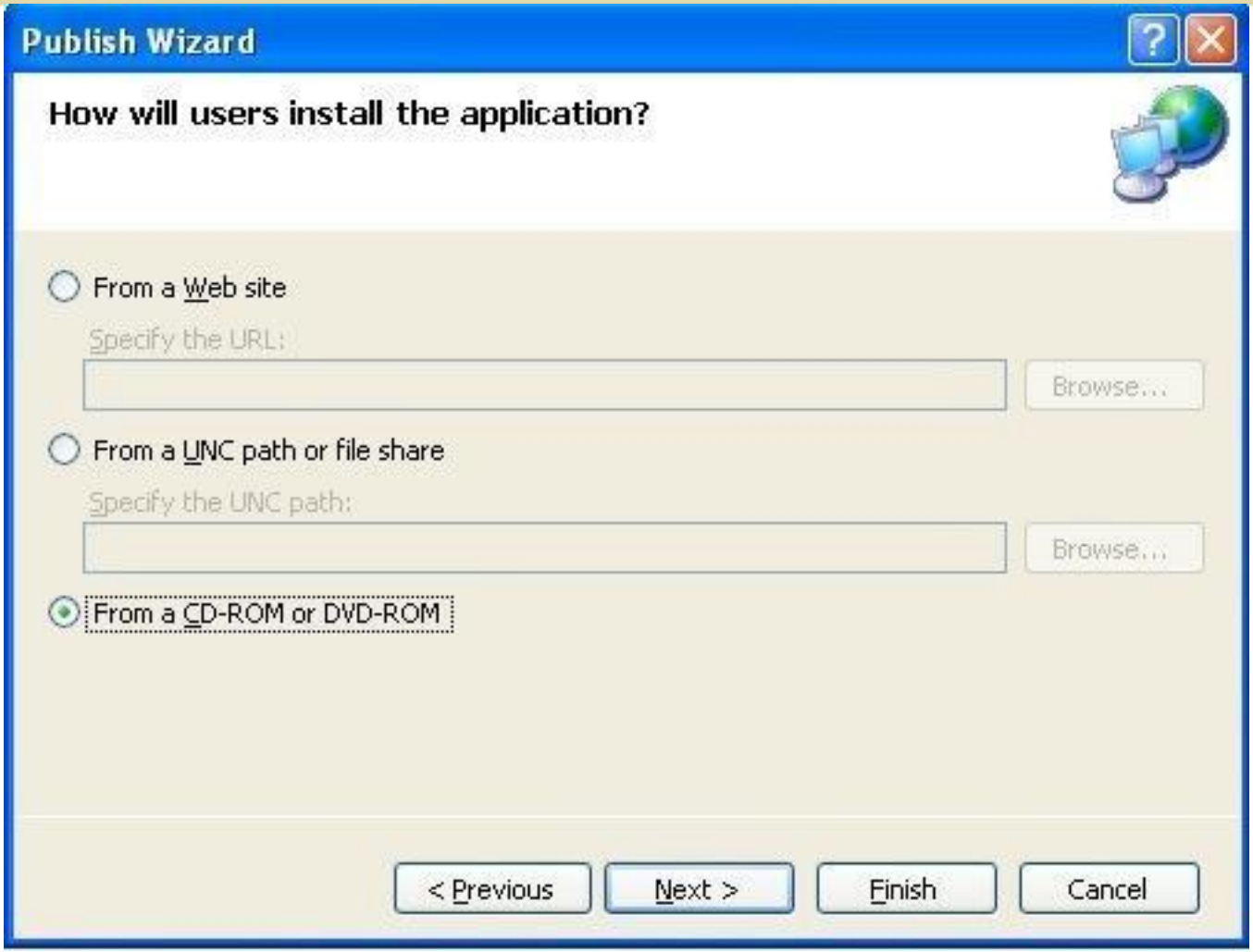
Now the application is packaged and is ready for distribution.

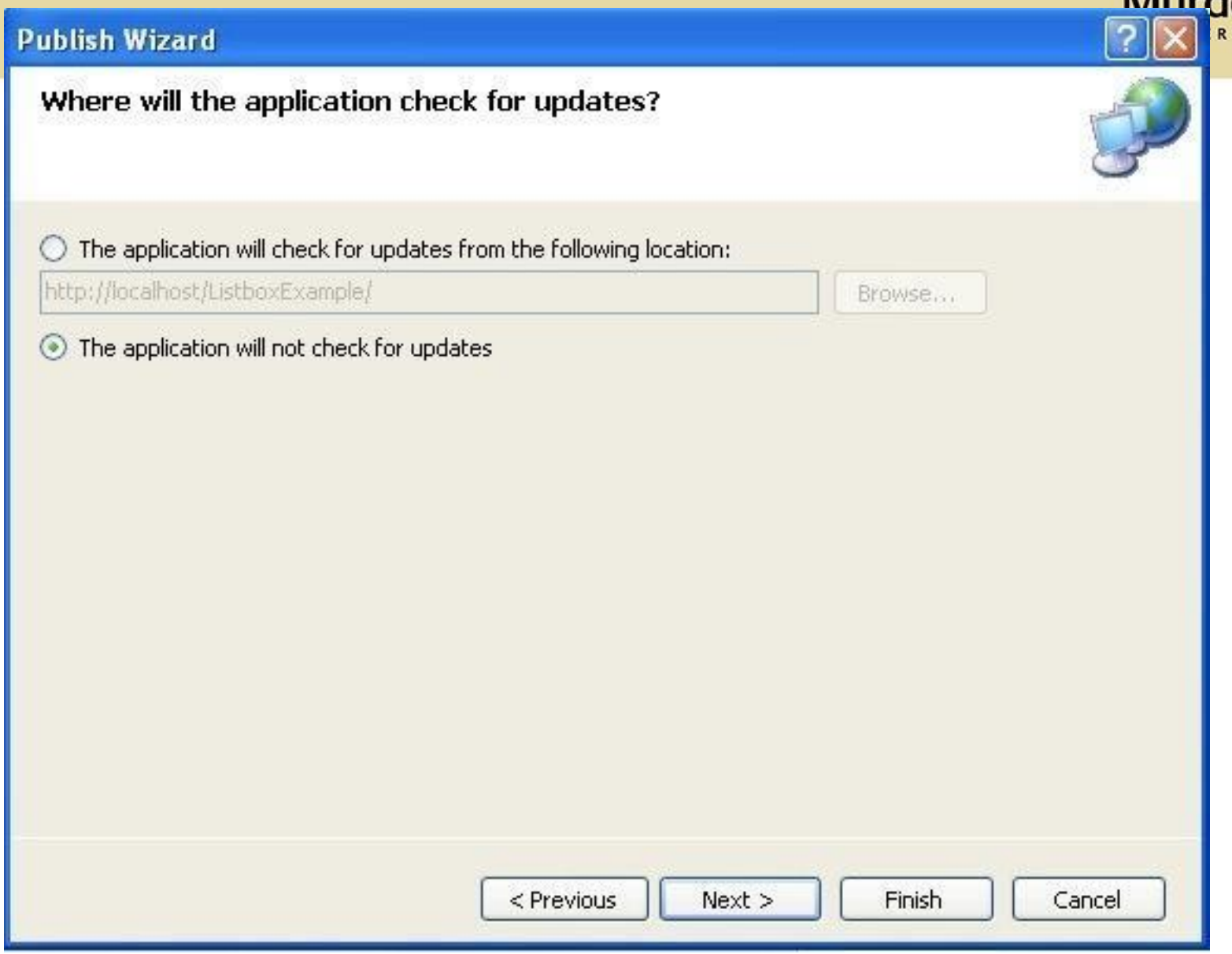
Copy the files in the temporary directory to a CD or a USB drive for distribution.

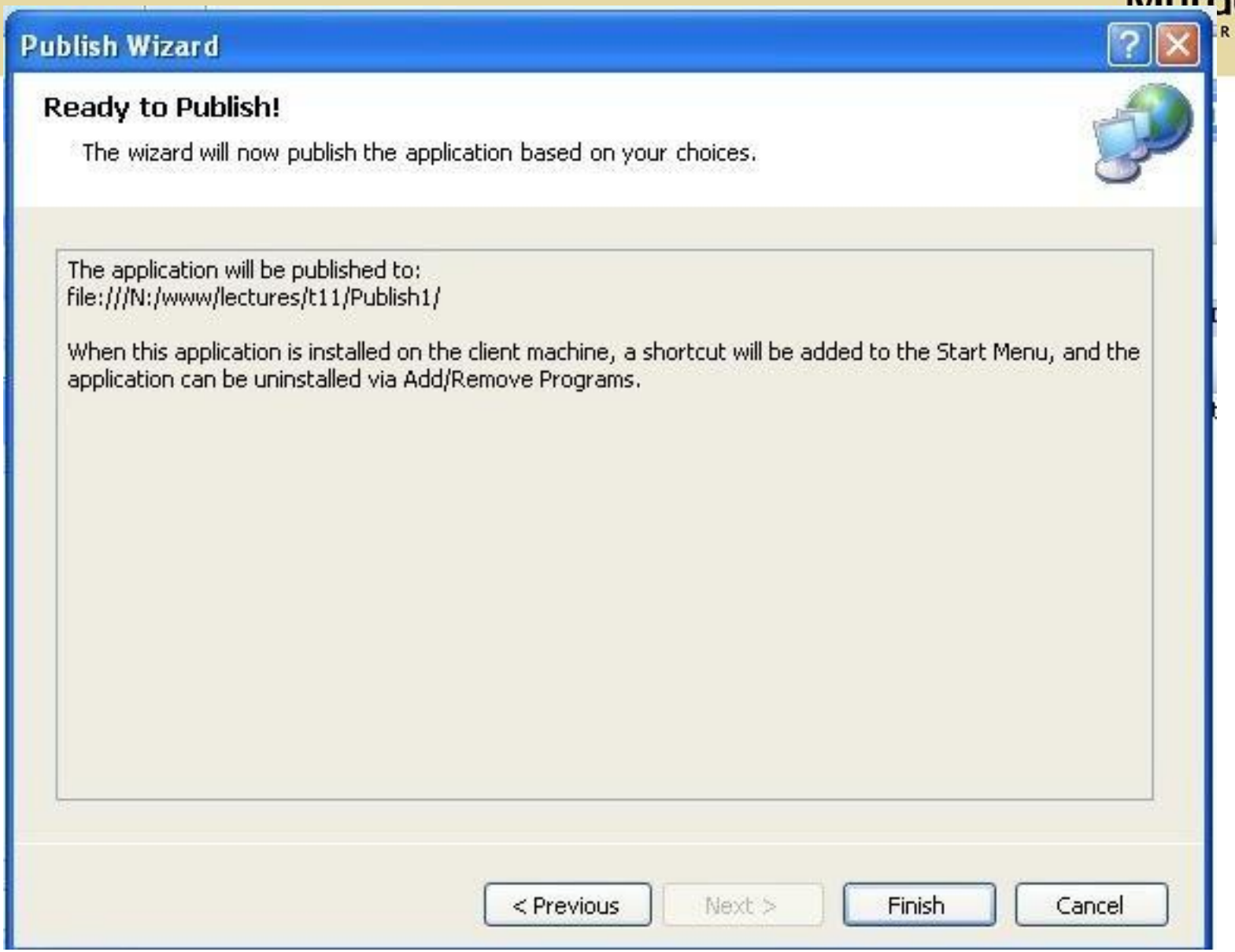
Murdoch
RSITY

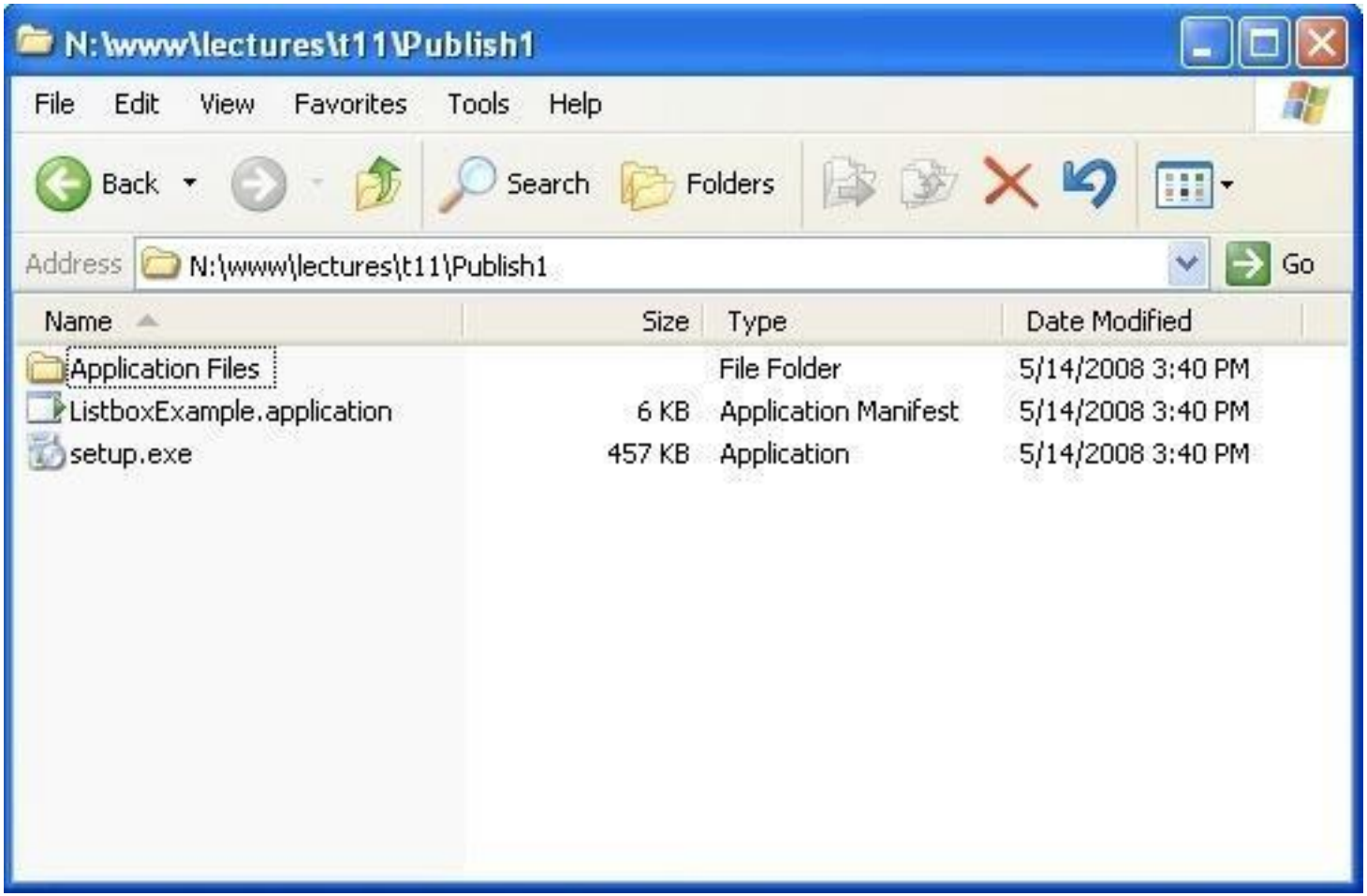


Murdoch UNIVERSITY




Publish Wizard
Where will the application check for updates?
The application will check for updates from the following location:
http://localhost/ListboxExample/
Browse...
The application will not check for updates
< Previous
Next >
Finish
Cancel


Publish Wizard
Ready to Publish!
The wizard will now publish the application based on your choices.
The application will be published to:
file:///N:/www/lectures/t11/Publish1/
When this application is installed on the client machine, a shortcut will be added to the Start Menu, and the application can be uninstalled via Add/Remove Programs.
< Previous
Next >
Finish
Cancel


N:\www\lectures\t11\Publish1
File Edit View Favorites Tools Help
Back
Search
Folders
Address
N:\www\lectures\t11\Publish1
Go
Name
Size
Type
Date Modified
Application Files
File Folder
5/14/2008 3:40 PM
ListboxExample.application
6 KB
Application Manifest
5/14/2008 3:40 PM
setup.exe
457 KB
Application
5/14/2008 3:40 PM

Use Property Dialog

Murdoch UNIVERSITY

An application can also be packaged using the properties page (not Properties window) from the Solution Explorer window:

Right-click the project from Solution Explorer window. Select Properties.

The Properties page pops up. Select “Publish” menu on the left.

Enter the necessary information.

Click “Option” to enter the name of your application such as “My listbox” and company name such ICT365. Your shortcut in the Start menu will be “ICT365=>My listbox”.

Enter the name of your deployment html page such as “publish.htm”

ListboxExample - Microsoft Visual C# 2008 Express Edition

File Edit View Project Build Debug Data Tools Window Help

ListboxExample* | Form1.cs [Design]

- Application
- Build
- Build Events
- Debug
- Resources
- Settings
- Reference Paths
- Signing
- Security
- Publish

Publish Location

Publishing Folder Location (web site, ftp server, or file path):

N:\www\lectures\t11\Publish2\ ...

Installation Folder URL (if different than above):

...

Install Mode and Settings

- ○ The application is available online only
- ◉ The application is available offline as well (launchable from Start menu)

Application Files...

Prerequisites...

Updates...

Options...

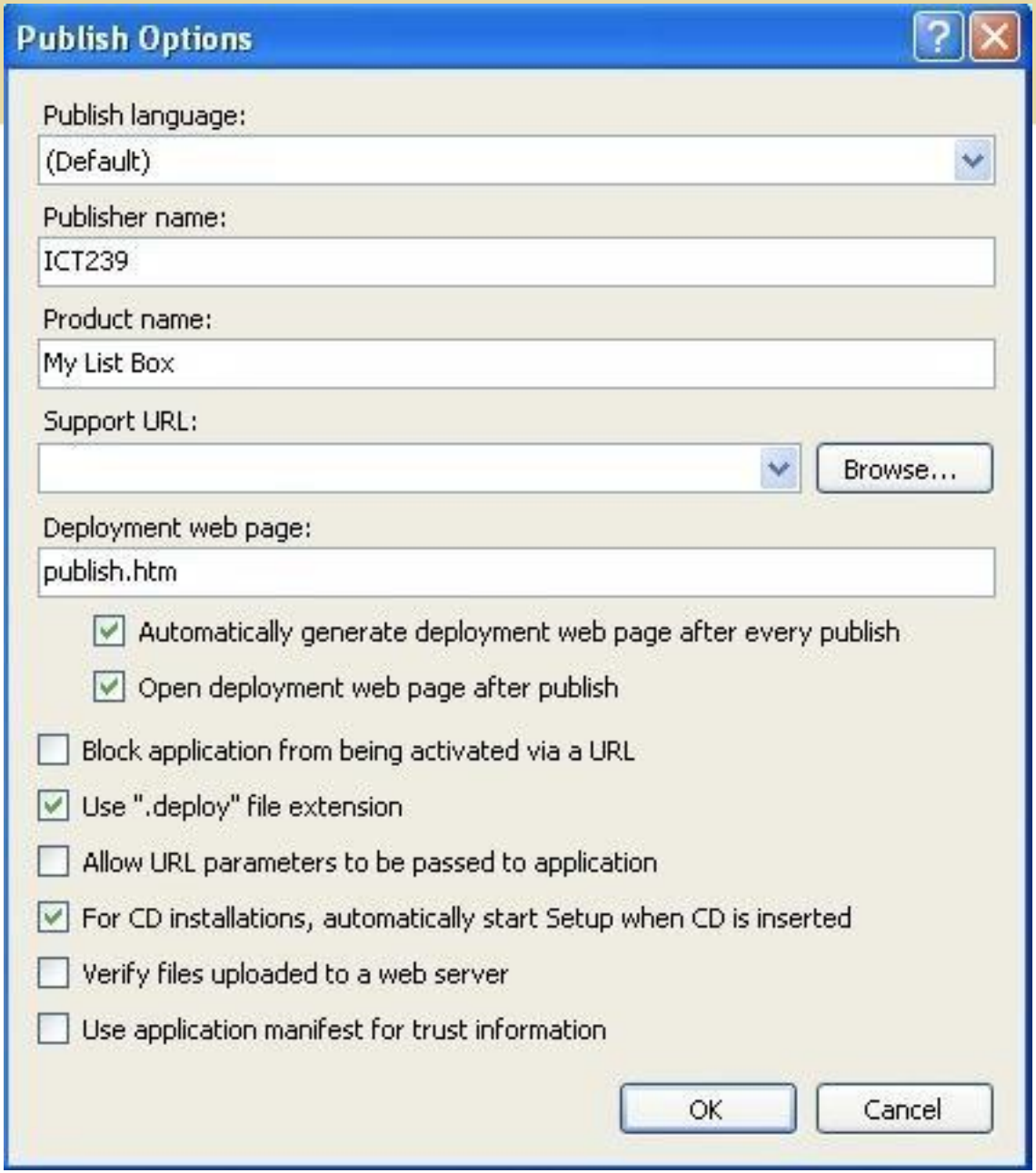
Publish Version

Major:	Minor:	Build:	Revision:
1	0	0	2

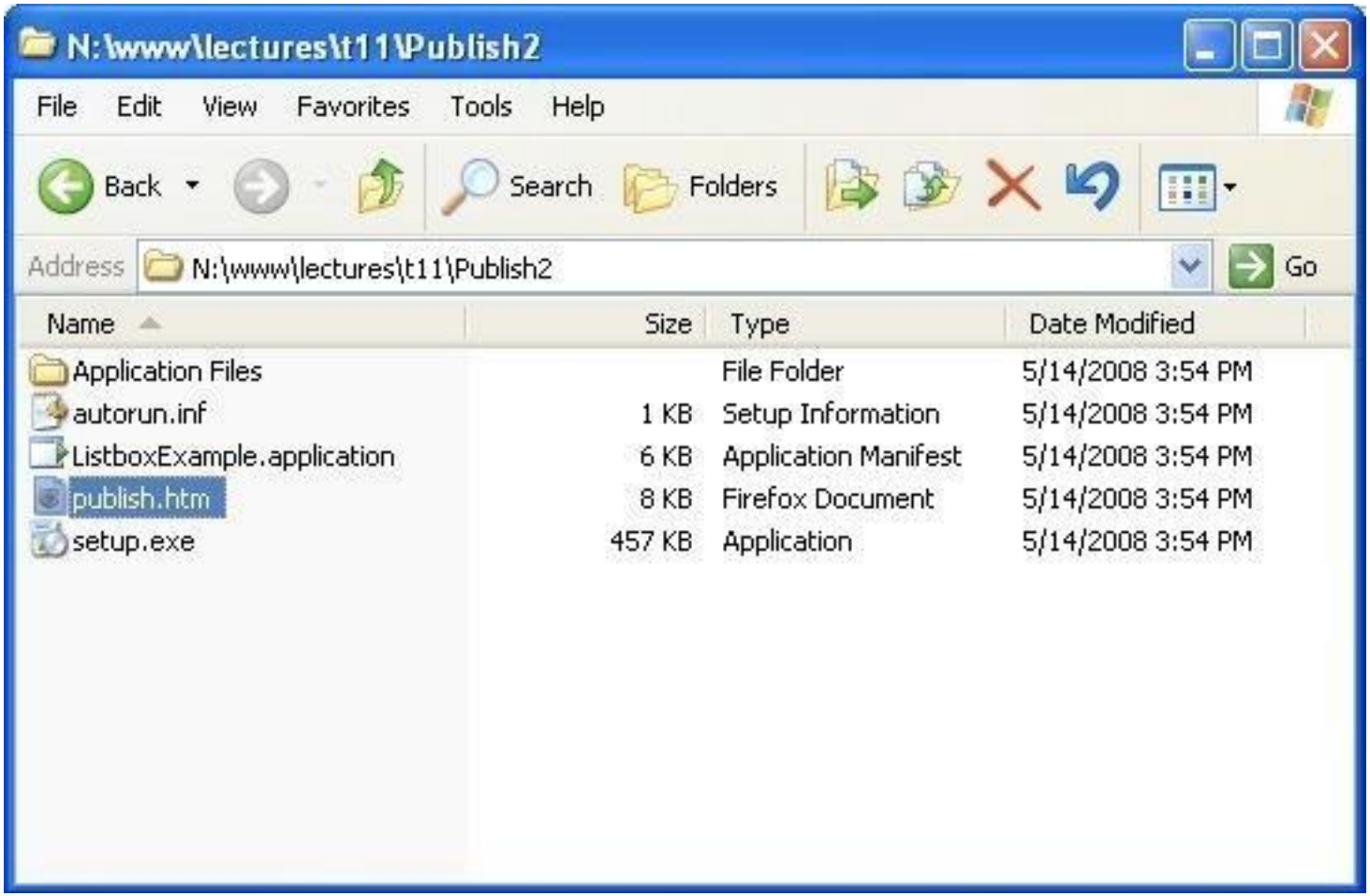
☑ Automatically increment revision with each publish

Publish Wizard... Publish Now

Publish succeeded


Publish Options
Publish language:
(Default)
Publisher name:
ICT239
Product name:
My List Box
Support URL:
Browse...
Deployment web page:
publish.htm
Automatically generate deployment web page after every publish
Open deployment web page after publish
Block application from being activated via a URL
Use ".deploy" file extension
Allow URL parameters to be passed to application
For CD installations, automatically start Setup when CD is inserted
Verify files uploaded to a web server
Use application manifest for trust information
OK
Cancel

Murdoch UNIVERSITY



Deployment From Web Only

Murdoch UNIVERSITY

Our next example demonstrates how to deploy an application from a website.

The application can only be run from the website. The application is not installed on the local machine.

Deployment Steps

Murdoch UNIVERSITY

Right-click the project from Solution Explorer window. Select Properties. The Properties page is displayed.

Enter the deployment directory. In our example, we use directory "N:\www\lectures\t11\Publish3".

Enter the deployment website url, eg:

This link is not active

http://www.it.murdoch.edu.au/units/ICT365/Test/MyListbox

Click "The application is available online only".

You can enter product name such as "My Listbox" and company name such as "ICT365" from Option menu.

Enter the name of the deployment web page, such as "run.htm".

Click "Publish Now". The application package will be created in the deployment directory "N:\www\lectures\t11\Publish3\".

Copy all files from the deployment directory to the deployment website.

ListboxExample - Microsoft Visual C# 2008 Express Edition

File Edit View Project Build Debug Data Tools Window Help

Form1.cs [Design] | ListboxExample*

Application
Build
Build Events
Debug
Resources
Settings
Reference Paths
Signing
Security
Publish*

Publish Location

Publishing Folder Location (web site, ftp server, or file path):

N:\www\lectures\t11\Publish3\ ...

Installation Folder URL (if different than above):

http://www.it.murdoch.edu.au/units/ICT239/Test/MyListbox/ ...

Install Mode and Settings

- (•) The application is available online only
- ( ) The application is available offline as well (launchable from Start menu)

Application Files...

Prerequisites...

Updates...

Options...

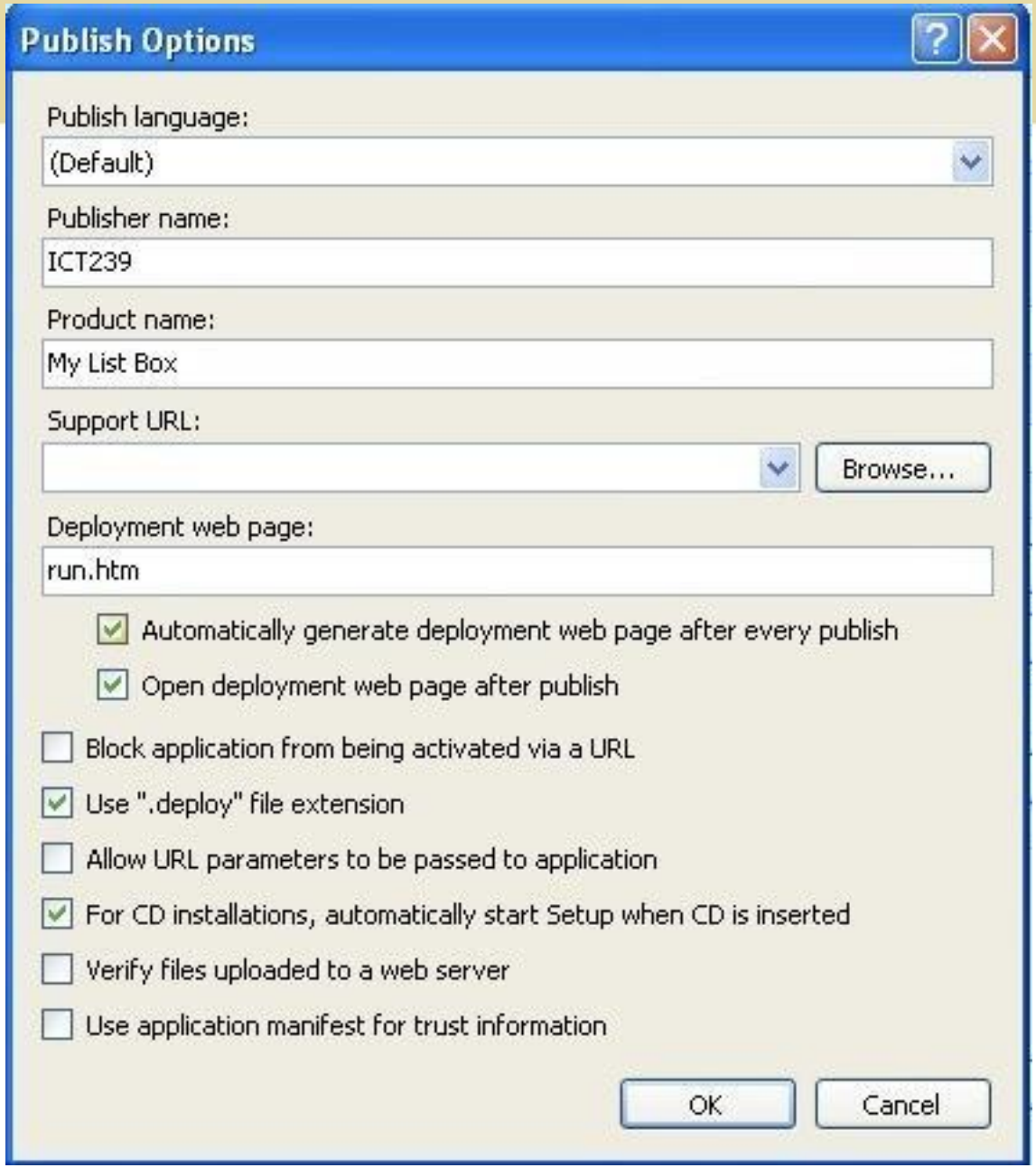
Publish Version

Major:	Minor:	Build:	Revision:
1	0	0	7

[x] Automatically increment revision with each publish

Publish Wizard... Publish Now

Publish succeeded


Publish Options
Publish language:
(Default)
Publisher name:
ICT239
Product name:
My List Box
Support URL:
Browse...
Deployment web page:
run.htm
Automatically generate deployment web page after every publish
Open deployment web page after publish
Block application from being activated via a URL
Use ".deploy" file extension
Allow URL parameters to be passed to application
For CD installations, automatically start Setup when CD is inserted
Verify files uploaded to a web server
Use application manifest for trust information
OK
Cancel

Run Application

Murdoch UNIVERSITY

To run the application, enter the following url in a web browser:

http://www.it.murdoch.edu.au/units/ICT365/Test/MyListbox/run.htm

Then click "Run" button in the web page to launch the application.

The application will be retrieved from the website to the local machine to run.

My List Box

http://www.it.murdoch.edu.au

Google

My List Box

ICT239
My List Box

Name: My List Box

Version: 1.0.0.7

Publisher: ICT239

The following prerequisites are required:

- Windows Installer 3.1
- .NET Framework 3.5

If these components are already installed, you can launch the application now. Otherwise, click the button below to install the prerequisites and run the application.

Run

ClickOnce and .NET Framework Resources

Limitations of ClickOnce

Murdoch UNIVERSITY

ClickOnce application does not use Registry. The application is installed in a completely separate place under the logon user's profile. It does not share files with other application.

It is good for thin client applications. For applications that need to share libraries, it may be better to use Microsoft Installer to create application package.

Summary

IL/MSIL/CIL - IL code is a CPU independent partially compiled code. It's partially compiled because we do not know in what kind of environment .NET code will run and on runtime IL Code will compile to machine code using the environmental properties (CPU, OS, machine configuration, etc).

ILDASM - This is a tool provided by Visual Studio to view IL code. To run ILDASM, we have to select option "Visual Studio Command Prompt" from "Visual Studio Tools" and type ildasm. It will open the ildasm tool where we can open any *exe/dll.ildasm* tool read the assembly by reflection and it is showing us various properties, methods which our assembly has. Here, we can see IL code of any method/property by clicking on that.

CLR - CLR is the heart of the .NET framework and it does 4 primary important things:

Garbage collection

CAS (Code Access Security)

CV (Code Verification)

IL to Native translation

CTS - CTS ensures that data types defined in two different languages get compiled to a common data type. This is useful because there may be situations when we want code in one language to be called in other language.

We can see a practical demonstration of CTS by creating the same application in C# and VB.NET and then compare the IL code of both applications. Here, the datatype of both IL code is same.

CLS - CLS is a subset of CTS. CLS is a set of rules or guidelines. When any programming language adheres to these set of rules, it can be consumed by any .NET language.CTS.

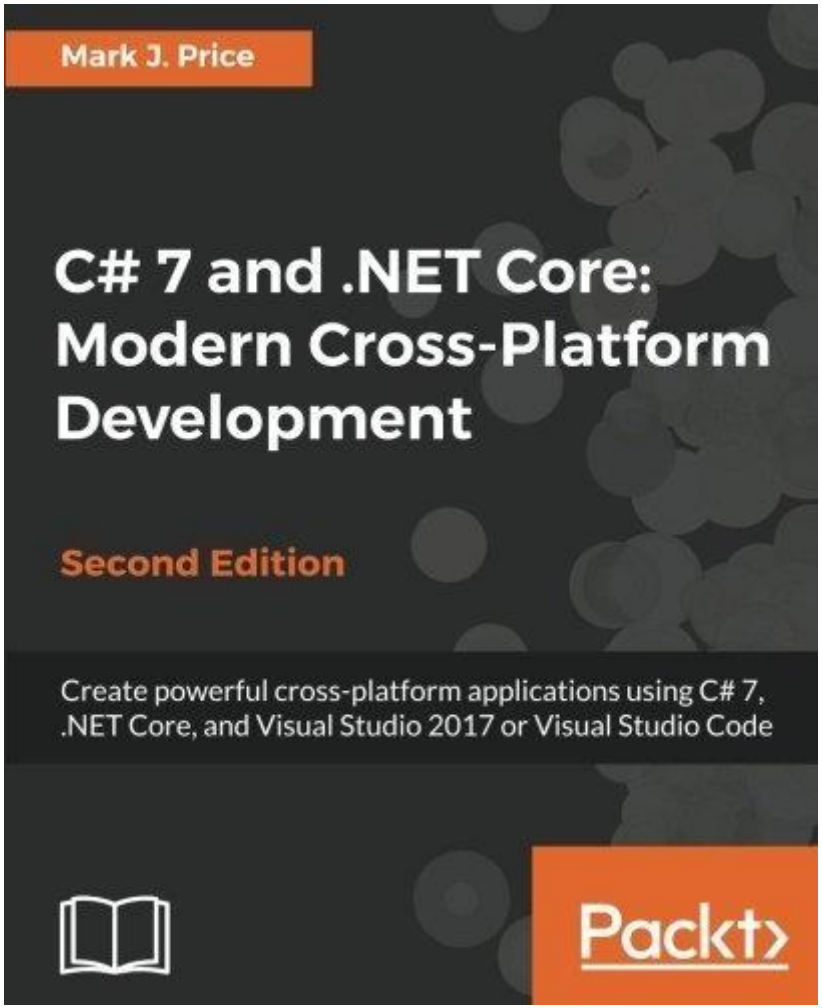
JIT - JIT compiles the IL code to Machine code just before execution and then saves this transaction in memory.

REFERENCES: Assemblies, deployment

Reading/ reference

http://prospero.murdoch.edu.au/record=b2962782~S1

Chapter 16. Packaging and Deploying Your Code Cross-Platform

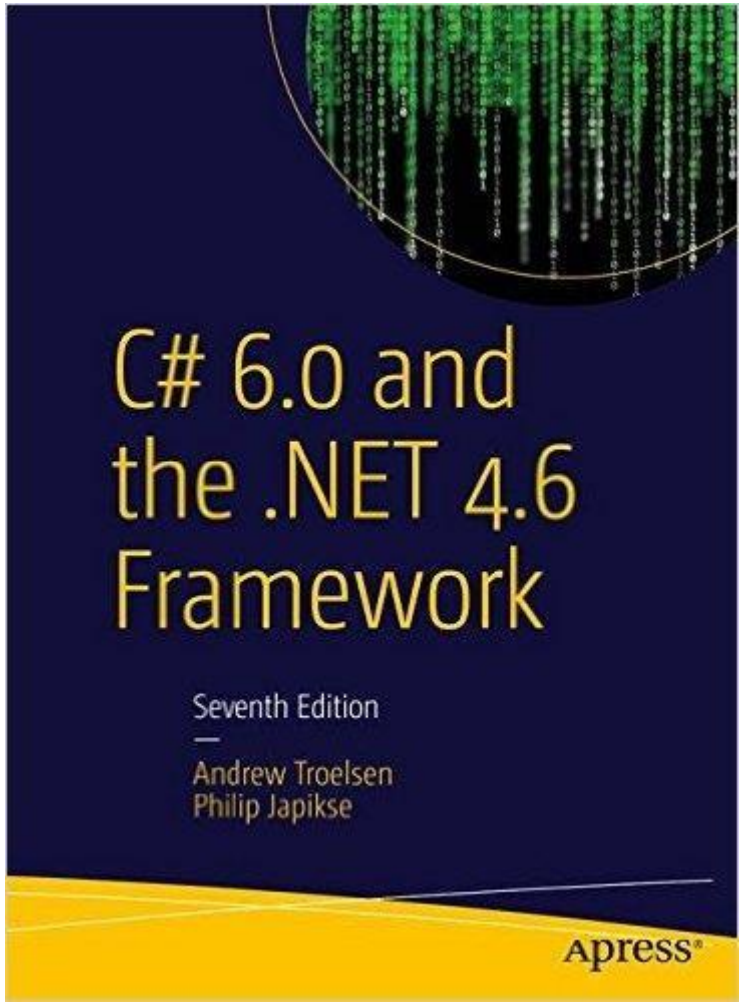


Reading/ reference

http://prospero.murdoch.edu.au/record=b2962780~S1

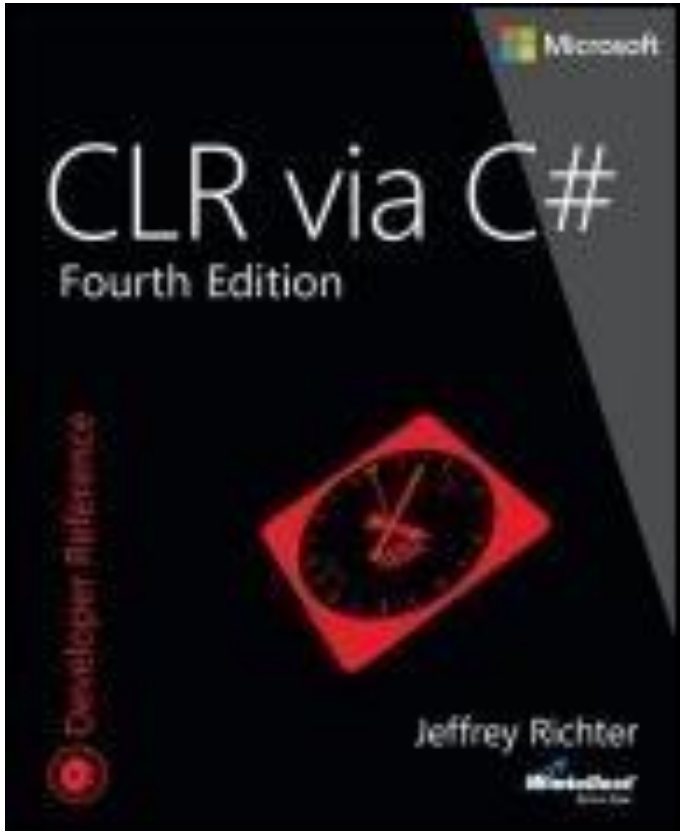
Murdoch UNIVERSITY

Chapter: Understanding CIL and the Role of Dynamic Assemblies



Reading/ reference

Chapter Assembly Loading and Reflection



REFERENCES: CLR, CIL

Reading/ reference

http://prospero.murdoch.edu.au/record=b2962780~S1

Murdoch UNIVERSITY

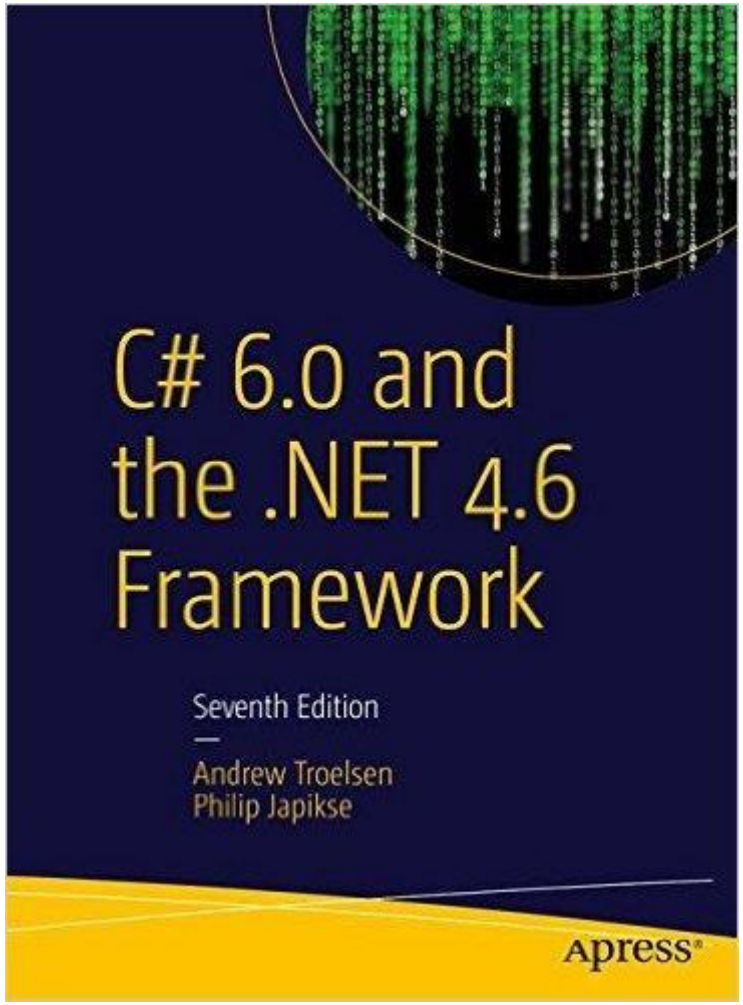
Chapter: Understanding Object Lifetime

Chapter: Building and Configuring Class Libraries

Chapter: Type Reflection, Late Binding, and Attribute-Based Programming

Chapter: Dynamic Types and the Dynamic Language Runtime

Chapter: Processes, AppDomains, and Object Contexts



Reading/ reference

Chapter 1. The CLR's Execution Model

Chapter 2. Building, Packaging, Deploying, and Administering Applications and Types

Chapter 3. Shared Assemblies and Strongly Named Assemblies

Chapter 21. The Managed Heap and Garbage Collection

Chapter CLR Hosting and AppDomains