Chapter 1

.NET Framework

.NET Framework

Objectives

After completing this unit you will be able to:

- **Describe the minimal "what you need to know" about .NET to start programming in the .NET environment.**
- **Answer the high level question "What is .NET?"**
  - Outline the new programming platform and tools provided by .NET.
- **Present an overview of the .NET Framework.**
- **Describe the Common Language Runtime.**
  - Explain the concepts of Microsoft Intermediate Language, metadata and JIT compilation.
  - Describe the use of assemblies in the CLR and outline the assembly/module/type hierarchy.
  - Explain the role of types and the Common Type System (CTS) as the heart of the CLR.

.NET: What You Need To Know

- **A beautiful thing about .NET is that from a programmer's perspective you scarcely need to know anything about it to start writing programs for the .NET environment.**
  - You write a program in a high-level language such as C#, a compiler creates an executable (.EXE) file, and you run that EXE file.
- **Even very simple programs, if they are to do something interesting, such as perform output, will require that the program employ the services of library code.**
  - A large library, called the .NET Framework Class Library, comes with .NET, and you can use all of the services of this library in your programs.

.NET: What Is *Really* Happening

- **The EXE file that is created does not contain executable code, but rather code in Intermediate Language, or IL (sometimes called Microsoft Intermediate Language, or MSIL).**
  - In the Windows environment, this IL code is packaged up in a standard portable executable (PE) file format, so you will see the familiar EXE extension (or, if you are building a component, the DLL extension).
- **When you run the EXE, a special runtime environment (the Common Language Runtime, or CLR) is launched, and the IL instructions are executed by the CLR.**
  - Unlike some runtimes, where the IL would be interpreted each time it is executed, the CLR comes with a just-in-time (JIT) compiler, which translates the IL to native machine code the first time it is encountered.
  - Then, on subsequent calls, the code segment runs as native code.

.NET Programming in a Nutshell

1. Write your program in a high-level .NET language such as C#.
2. Compile your program into IL.
3. Run your IL program, which will launch the CLR to execute your IL, using its JIT to translate your program to native code as it executes.

Understanding .NET

- **The nice thing about a high-level programming language is that for the most part you do not need to be concerned with the platform on which the program executes.**
- **You can work with the abstractions provided by the language and with functions provided by libraries.**
- **Your appreciation of the C# programming language and its potential for creating great applications will be richer if you have a general understanding of .NET.**
- **The rest of this chapter is concerned with helping you to achieve such an understanding. We will address three broad topics:**
  - What Is Microsoft .NET?
  - .NET Framework
  - Common Language Runtime

What Is .NET?

- **Microsoft .NET is a new platform at a higher level than the operating system.**
  - Three years in the making before public announcement, .NET is a major investment by Microsoft.
  - .NET draws on many important ideas, including XML, the concepts underlying Java, and COM.
- **Microsoft .NET provides:**
  - A robust runtime platform, the Common Language Runtime
  - Multiple language development
  - An extensible programming model, the .NET Framework, which includes a very large class library of reusable code available from multiple languages
  - A networking infrastructure built on top of Internet standards that supports a high level of communication among applications
  - A new mechanism of application delivery, the Web Service, that supports the concept of an application as a service
  - Powerful development tools, including tools to greatly simplify both Windows programming and Web programming

A New Programming Platform

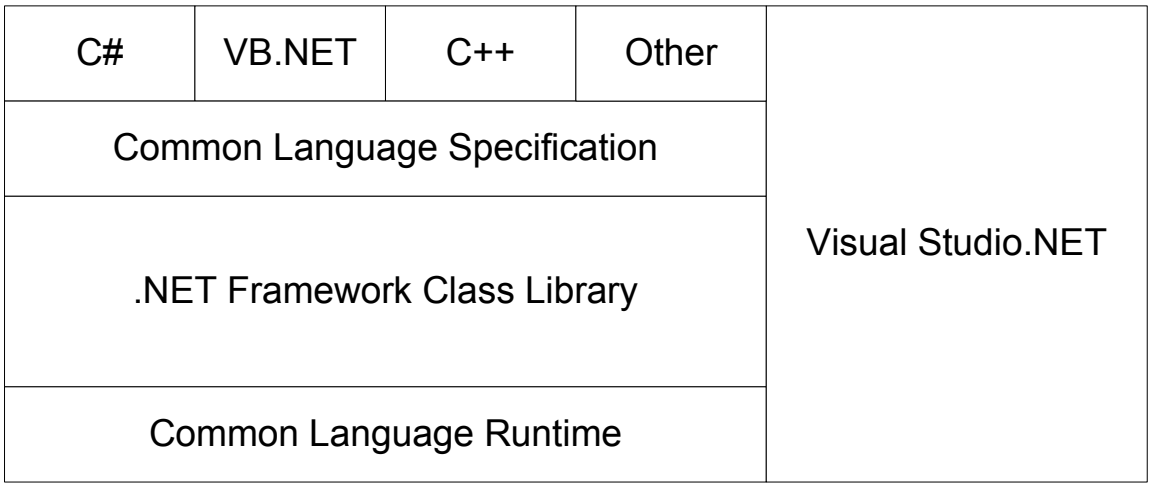
- **.NET provides a new programming platform at a higher level than the operating system.**
- **This level of abstractions has many advantages:**
  - Safety and security checking can be done, providing more robust operation.
  - The higher level platform is much easier to program than at the lower level of the Win32 API or COM.
  - Potentially the whole platform can be implemented on many different kinds of computers (as has been done with Java).
  - One class library is used by all the languages
  - Languages can interoperate with each other.

Multiple Language Development

- **As its name suggests, the CLR supports many programming languages.**
  - A "managed code" compiler must be implemented.
- **Microsoft itself has implemented compilers for managed C++, Visual Basic.NET, JScript and the new language C#.**
- **Well over a dozen other languages are being implemented by third parties, among them COBOL by Fujitsu and Perl by ActiveState.**
  - PerlNET is a practical tool that lets you call .NET classes from Perl programs, and vice versa.
- **Programmers do not need to be retrained in a new language in order to gain the benefits of .NET.**

.NET Framework Overview

- **There are five principal parts to .NET Framework:**
  - Common Language Runtime
  - .NET Framework Class Library
  - Common Language Specification
  - .NET Languages
  - Visual Studio.NET

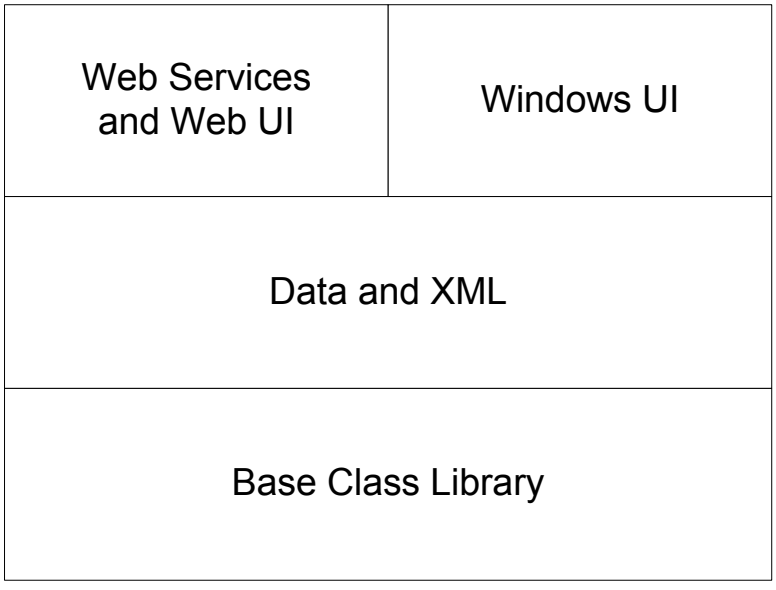


Common Language Runtime

- **A runtime provides services to executing programs.**
- **Traditionally there are different runtimes for different programming environments.**
  - Examples of runtimes include the standard C library, MFC, the Visual Basic runtime and the Java Virtual Machine.
- **The runtime environment provided by .NET is called the *Common Language Runtime* or CLR.**
- **The CLR manages the execution of code and provides useful services.**
- **The services of the CLR are exposed through programming languages.**
  - The syntax for these services varies from language to language.
- **But the underlying engine providing the services is the same.**
- **Not all languages expose all the features of the CLR.**
- **The language with the best mapping to the CLR is the new language C# ("C sharp").**
- **The CLR will be examined in greater detail in the next section.**

.NET Framework Class Library

- **The .NET Framework class library is huge, comprising over 2500 classes.**
  - All this functionality is equally available to all the .NET languages.
- **The library consists of four main parts:**
  - Base class library (includes networking, security, diagnostics, I/O and other operating system type services)
  - Data and XML classes
  - Web services and web UI
  - Windows UI



Common Language Specification

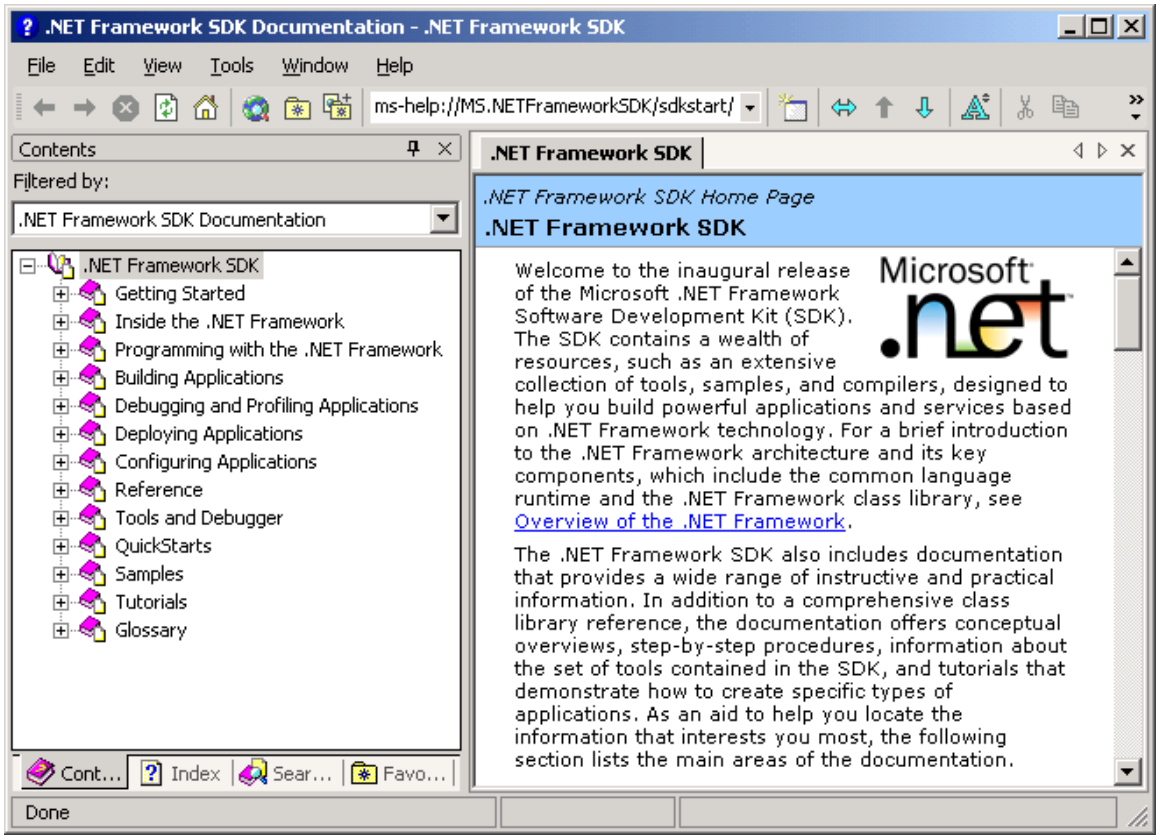
- **An important goal of the .NET Framework is to support multiple languages.**
- **But all languages are not created equal, so it is important to agree upon a common subset that *all languages* will support.**
- **The Common Language Specification (CLS) is an agreement among language designers and class library designers about those features and usage conventions that can be relied upon.**
- **CLS rules apply to public features that are visible outside the assembly where they are defined.**
- **A typical CLS rule is that public names should not rely on case for uniqueness, because some languages are not case sensitive.**
- **The complete rules are contained in the ECMA specifications.**

Languages in .NET

- **A language is a CLS-compliant *consumer* if it can *use* any CLS-compliant type, i.e. call methods, create instances of types, etc.**
- **A language is a CLS-compliant *extender* if it is a consumer and can also extend any CLS-compliant base class, implement any CLS-compliant interface, etc.**
- **Microsoft itself is providing four CLS-compliant languages.**
  - C#, Visual Basic .NET and Visual C++ .NET are all extenders.
  - JScript .NET is a consumer.
- **Third parties are providing additional languages (over a dozen so far).**
  - ActiveState is implementing Perl and Python (http://aspn.activestate.com/ASPN).
  - Fujitsu is implementing COBOL (http://www.adtools.com/info/whitepaper/net.html).
  - Oberon Microsystems is implementing Pascal (http://www.oberon.ch/)
  - QKS is implementing Smalltalk (http://www.qks.com/)
  - For other third-party offerings, see http://msdn.microsoft.com/net/thirdparty/default.asp

.NET Framework SDK

- **The .NET Framework SDK provides a complete set of tools, documentation and sample programs for building .NET applications.**
  - The tools are invoked from the command line.
  - The SDK is available on one CD (and is also included with a multiple CD distribution of Visual Studio.NET).
- **The .NET Framework SDK provides extensive online documentation.**



.NET Framework Class Library

- **Modern programming relies heavily on reusable code provided in libraries.**
- **Object-oriented languages facilitate the creation of class libraries, which are flexible, have a good degree of abstraction, and are extensible by adding new classes and basing new classes on existing ones, "inheriting" existing functionality.**
- **The .NET Framework Class Library provides over 2500 classes of reusable code, which can be called by all the .NET languages.**
- **The class library is extensible, and new classes can inherit from existing classes, even ones implemented in a different language.**
- **.NET Classes support Windows programming, web programming, database programming, XML, interoperability with COM and Win32, and many, many important features.**

Development Tools

- **A practical key to success in software is effective tools.**
- **Microsoft has long had great tools, including Visual C++ and Visual Basic.**
- **With .NET they have combined their development tools into a single integrated environment called *Visual Studio .NET***
  - VS.NET provides a very high degree of functionality for creating applications in all the languages supported by .NET.
  - You can do multiple language debugging, etc.
  - VS.NET has many kinds of *designers* for forms, databases and other software elements.
- **As with the languages themselves, third parties can provide extension to Visual Studio .NET, creating a seamless development environment for their language that interoperates with the other .NET language.**
- **The tool set includes extensive support for building web applications and web services.**
- **There is also great support for database applications.**

Common Language Runtime

- **In this section we go more deeply into the structure of .NET by examining the CLR. At the end of this section you will be able to**
  - Outline the design goals of the Common Language Runtime (CLR).
  - Discuss the rationale for using managed code and a runtime.
  - Explain the concepts of Microsoft Intermediate Language, metadata and JIT compilation.
  - Describe the use of assemblies in the CLR and outline the assembly/module/type hierarchy.
  - Explain the role of types and the Common Type System (CTS) as the heart of the CLR.
  - Explain the role of managed data and garbage collection.

Design Goals of the CLR

- **The Common Language Runtime has the following design goals:**
  - Simplify application development
  - Support multiple programming languages
  - Provide a safe and reliable execution environment
  - Simplify deployment and administration
  - Provide good performance and scalability

Why Use a CLR?

- **Why did Microsoft create a CLR for .NET?**
- **Let's look at how far the goals just discussed could have been achieved without a CLR, focusing on the two main goals of:**
  - Safety
  - Performance
- **Basically there are two philosophies:**
  - Compile-time checking and fast native code at runtime.
  - Runtime checking.
- **Without a CLR, we must rely on the compiler to achieve safety.**
- **This places a high burden on the compiler.**
  - Typically there are many compilers for system, including third-party compilers.
  - It is not robust to trust that every compiler will adequately perform all safety checking.
  - Not every language has features supporting adequate safety checking.
  - Compilation speed is slow with complex compilation.
  - Compiler cannot optimize code based on enhanced instructions available on some platforms but not others.

Intermediate Language

- **So we want a runtime, how do we design it?**
- **One extreme is to use an *interpreter* and not a compiler at all.**
  - *All* the work is done at runtime.
  - We have safety and fast "builds", but runtime performance is very slow.
- **Modern systems divide the load between the front-end compiler and the back-end runtime.**
- **The front-end compiler does all the checking it can do and generates an *intermediate language*. Examples:**
  - P-code for Pascal
  - Bytecode for Java
- **The runtime does further verification based on the actual runtime characteristics, including security checking.**
- **With "just-in-time" (JIT) compilation, native code can be generated when needed, and subsequently reused.**
  - Runtime performance becomes much better.
  - The native code generated by the runtime can be more efficient, because the runtime knows the precise characteristics of the target machine.

Microsoft Intermediate Language

- **All managed code compilers for Microsoft .NET generate Microsoft Intermediate Language (MSIL).**
  - MSIL is machine-independent and can be efficiently translated into native code.
- **MSIL has a wide variety of instructions:**
  - Standard operations such as load, store, arithmetic and logic, branch, etc.
  - Calling methods on objects
  - Exceptions
- **Before executing on a CPU, MSIL must be translated by a *just-in-time* compiler.**
- **There is a JIT compiler for each machine architecture supported.**
  - The same MSIL will run on any supported machine.

Metadata

- **Besides generating MSIL, a managed code compiler emits *metadata*.**
- **Metadata contains very complete information about the code module and all the types within it:**
  - Version and locale information
  - All the types
  - Details about each type, including name, visibility, etc.
  - Details about the members of each type, such as methods, the signatures of methods, etc.
- **Metadata is the "glue" that binds together the executing code, the CLR, and tools such as compilers, debuggers, browsers, etc.**
- **On Windows MSIL and metadata are packaged together in a standard Windows portable executable file (PE).**

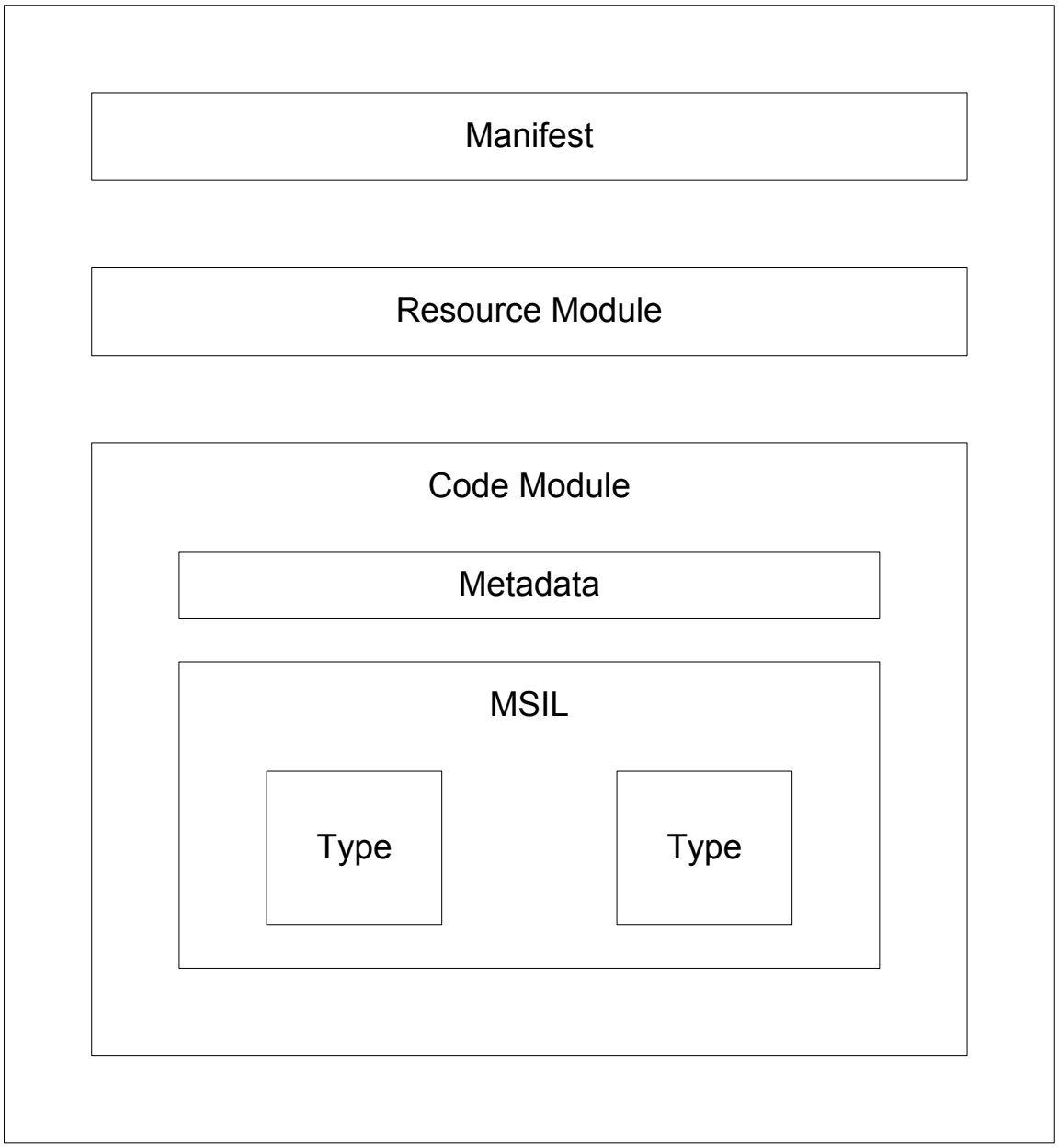
JIT Compilation

- **Before executing on the target machine, MSIL is translated by a just-in-time (JIT) compiler to native code.**
- **Some code typically will never be executed during a program run.**
  - Hence it may be more efficient to translate MSIL as needed during execution, storing the native code for reuse.
- **When a type is loaded, the loader attaches a stub to each method of the type.**
  - On the first call the stub passes control to the JIT, which translates to native code and modifies the stub to save the address of the translated native code.
  - On subsequent calls to the method transfer is then made directly to the native code.
- **As part of JIT compilation code goes through a verification process.**
  - Type safety is verified, using both the MSIL and metadata.
  - Security restrictions are checked.

Assemblies

- **An assembly is a grouping of types and resources that work together as a logical unit.**
  - An assembly can be thought of as a logical DLL or EXE.
  - An assembly consists of one or more physical files, called **modules**, which may be PE code files or resources (such as bitmaps).
- **An assembly holds three kinds of information:**
  - MSIL implementing one or more types
  - Metadata
  - A **manifest** describing how the elements in the assembly relate to each other and to external elements
- **An assembly forms the boundary for:**
  - Security
  - Deployment
  - Type resolution
  - Versioning

Assembly Hierarchy



Types

- ***Types*** **are at the heart of the programming model for the CLR.**
- **A Type is basically a Class in most object-oriented programming languages, providing an abstraction of data and behavior, grouped together.**
- **A Type in the CLR contains:**
  - Fields (data members)
  - Methods
  - Properties
  - Events
- **There are also built in primitive types, such as integer and floating point numeric types, string, etc.**
- **In the CLR there are no functions outside of types, but all behavior is provided via methods or other members.**
- **We will discuss types under the guise of classes when we cover C#.**

Common Type System

- **The *Common Type System* (CTS) provides a wide range of types and operations that are found in many programming languages.**
- **The CTS is shared by the CLR and by compilers and other tools.**
- **The CTS provides a framework for cross-language integration and addresses a number of issues:**
  - Similar but subtly different type (e.g. **Integer** is 16 bits in VB6 but **int** is 32 bits in C++, strings in VB6 are represented as BSTRs and in C++ as **char** pointers or a **string** class of some sort, etc.).
  - Limited code reuse (e.g. you can't define a new type in one language and import it into another language)
  - Inconsistent object models
- **Not all CTS types are available in all languages.**
  - The Common Language Specification (CLS) establishes rules that must be followed for cross language integration, including which types *must* be supported by a CLS-compliant language.
- **Built-in types can be accessed through the *System* class in the Base Class Library (BCL) and through reserved keywords in the .NET languages.**

Summary

- **C# does not exist in isolation, but has a close connection with the underlying .NET Framework.**
  - In this introductory chapter, we began by telling you just enough about the .NET Framework so that, if you so desired, you could proceed directly to Chapter 2 and start learning C#.
- **Microsoft .NET is a new platform at a higher level than the operating system that provides many capabilities for building and deploying both standard applications and new web-based ones.**
- **The .NET Framework includes the Common Language Runtime (CLR), the .NET Framework class library, the Common Language Specification (CLS), the .NET languages, and Visual Studio.NET.**
- **The CLR manages the execution of code and provides useful services.**
  - .NET compilers translate source code into MSIL, which is translated at runtime into native code by a JIT compiler.
  - An assembly is a grouping of types and resources that work together as a logical unit.
  - Types and the Common Type System are the heart of the CLR.