FIREBRAND

# Microsoft

# MCSD: Windows Store Style Apps Using C# Certification

# 70-483: Programming in C# Courseware

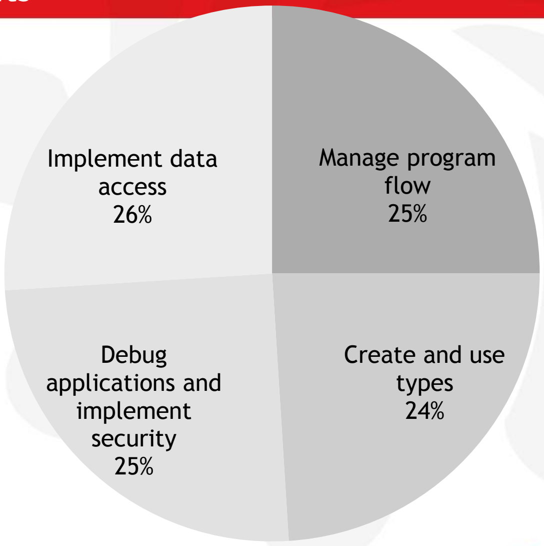
Version 1.0

www.firebrandtraining.com

Module 1
Review of Visual C# Syntax

## Course and Exam Contents

- 42 questions
- 130 minutes



MSDN study links for exam 70-483: Programming in C#
`http://www.jayway.com/2012/08/15/msdn-study-links-for-exam-70-483-programming-in-c/`

Review of Visual C# Syntax

## Contents

**Exam Topic: Implement program flow**
- ❑ Program decisions by using switch statements (1-20), if/then (1-20), and operators (1-9)
- ❑ Iterate across collection and array items (1-21)
- ❑ Evaluate expressions (1-11)

**Exam Topic: Manipulate strings**
- ❑ Manipulate strings by using the StringBuilder, StringWriter, and StringReader classes (1-17)
- ❑ Search strings (1-17)
- ❑ Enumerate string methods (21)
- ❑ Format strings (35)

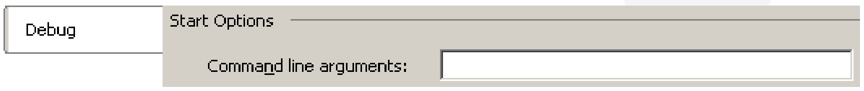
**Exam Topic: Consume types**
- ❑ Box or unbox to convert between value types (4-6)
- ❑ Cast types (32)
- ❑ Convert types (1-14)

**Exam Topic: Validate application input**
- ❑ Data collection types (1-8)
- ❑ Manage data integrity (1-17)
- ❑ Evaluate a regular expression to validate the input format (24)
- ❑ Use built-in functions to validate data type and content (1-16)

Labs and Demos

## Console Applications

- Text-based command line user interface
  - Many of the lab exercises use console applications
- If you start a Console app with debugging (F5), then Visual Studio closes the command prompt window automatically when it terminates, so instead:
  - Use Ctrl+F5 instead (leaves command prompt window open)
  - Add a Console.ReadLine() as last line
  - Set breakpoint on last line
- To pass command line arguments from Visual Studio
  - Project properties, Debug tab, Start Options section



Namespaces

## What Are They?

- A namespace is a logical container of type
- To import a namespace
  - Without importing the namespace, declarations can get long

```
System.Collections.Hashtable ht;
```

  - Importing a namespace is optional, but it simplifies declarations

```
using System.Collections;
...
Hashtable ht;
```

- To define a namespace

```
namespace MyCompany.Controls
{
  ...
}
```

Assemblies

## What Are They?

- An assembly is a container for everything the CLR needs to load and execute your code
- Assembly metadata and manifest
  - Name, assembly and file version, strong name, referenced assemblies, version of CLR to use, and so on
- Type metadata
  - Information about the types, their members, and so on
- Code
  - Intermediate Language (IL) code for methods
- Embedded Resources (optional)
  - Images, strings, JavaScript, and so on

Assemblies

## Command Line Tools

- csc.exe and vbc.exe
  - Language compilers; create .exe, .dll, and .netmodule files
- resgen.exe
  - Resource compiler turns .resx (XML) into .resources (binary)
- al.exe
  - Assembly linker combines metadata, .netmodules, resources
- sn.exe
  - Generate a strong name key pair (required for GAC deployment)
- gacutil.exe
  - Install or uninstall assembly in Global Assembly Cache (GAC)

Assemblies

## How Are They Related to Namespaces?

- To reference an assembly (actual container of type)
  - Required to use a type in that assembly
  - To use command line compiler instead of Visual Studio

```
csc /r:AnAssembly.dll mycode.cs
```

- To use Object Browser to explore relationship between assemblies and namespaces
  - For example, XmlDataDocument is in the System.Data assembly, but logically in the System.Xml namespace
  - To use XmlDataDocument, you must reference the System.Data assembly, and can import the System.Xml namespace
- Object Browser can show types grouped by container (i.e. assembly) or by namespace

Types
## What Are They?

- When a variable is declared you specify the type
  - Local to where they are declared e.g. inside an *if* or *try* block

```
Button       b;
Int32        i;
XmlNodeType  x;
IDisposable  r;
EventHandler e;
```

| Category | Type | Meaning |
|---|---|---|
| class | Button | Defines a *reference* type stored on the heap |
| struct | Int32 | Defines a *value* type stored on the stack |
| enum | XmlNodeType | Lookup of string constants stored as *integers* |
| interface | IDisposable | *Contract* that a type can implement |
| delegate | EventHandler | Type-safe *function pointer* |

Types (C# Programming Guide)
http://msdn.microsoft.com/library/ms173104.aspx

Types
## Members

| Member | Example | Meaning |
|---|---|---|
| **Field** | String. Empty | Used for data storage; can be passed by ref, unlike a property; can be read-only |
| Constant | Int32. MaxValue | Field that can never change value; or use a read-only field |
| Event | Button. Click | Field of type delegate; creates get/set methods to implement event handler functionality |
| **Method** | ToString | An action; can be called with delegate |
| Constructor | | Method executed when an instance is created |
| Property | Button. Width | Syntactic sugar for pair of methods to get and/or set a value; often has field as the backing store |
| Indexer | this[int] | Syntactic sugar; s[2] to access char in string |
| Operator | + | Syntactic sugar; + calls String.Concat |

## Types
## Type and Member Access and Other Modifiers

1.11

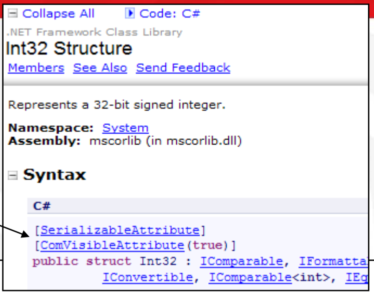
| Keyword | Visible outside type? |
|---|---|
| private | No; default for nested classes and structs |
| protected | Only to derived types |
| internal | Only to types in same assembly; default |
| protected internal | Only to derived types OR types in same assembly |
| public | Yes, everywhere; default for nested enums |

| Keyword | Meaning |
|---|---|
| static | One copy of field or method is shared by all type instances; access members via type |
| readonly | Field that can only be set in a constructor |

## Types
## How to Use the MSDN Library

1.12

- .NET Framework Class Library
  - Brief description
  - Namespace, Assembly
  - Syntax (inheritance, interfaces)
  - Remarks (good for exam study)
  - Examples
  - Inheritance Hierarchy
  - See Also, Members



Int32 Structure

Members See Also Send Feedback

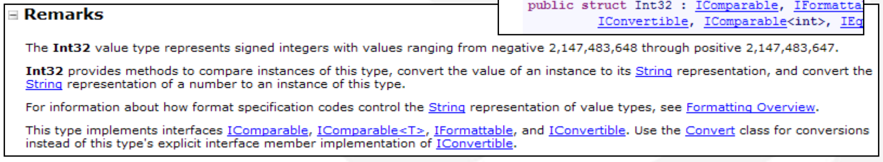
Represents a 32-bit signed integer.

**Namespace:** System
**Assembly:** mscorlib (in mscorlib.dll)

**Syntax**

C#

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public struct Int32 : IComparable, IFormatta
            IConvertible, IComparable<int>, IEq
```



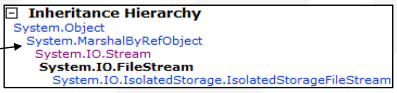
**Remarks**

The **Int32** value type represents signed integers with values ranging from negative 2,147,483,648 through positive 2,147,483,647.

**Int32** provides methods to compare instances of this type, convert the value of an instance to its String representation, and convert the String representation of a number to an instance of this type.

For information about how format specification codes control the String representation of value types, see Formatting Overview.

This type implements interfaces IComparable, IComparable<T>, IFormattable, and IConvertible. Use the Convert class for conversions instead of this type's explicit interface member implementation of IConvertible.



**Inheritance Hierarchy**

```
System.Object
  System.MarshalByRefObject
    System.IO.Stream
     System.IO.FileStream
       System.IO.IsolatedStorage.IsolatedStorageFileStream
```

System.Int32
`http://msdn.microsoft.com/en-us/library/system.int32.aspx`

Value Types

## Declaring Value Types

Some built-in value types

- `Int16, Int32, Int64, UInt32, Single, Double, Decimal`
- `Char, Boolean, DateTime`

```
int   a1, short b1;
Int32 a2, Int16 b2;
```

- Assignment copies the value stored on the stack
- U prefix indicates unsigned, exception is Byte/SByte

```
Console.WriteLine("UInt16 range is {0} to {1}",
  UInt16.MinValue, ushort.MaxValue); // 0 to 65,535
Console.WriteLine("Byte range is {0} to {1}",
  Byte.MinValue, byte.MaxValue); // 0 to 255
Console.WriteLine("SByte range is {0} to {1}",
  SByte.MinValue, sbyte.MaxValue); // -127 to 128
```

Classes and Structs (C# Programming Guide)
http://msdn.microsoft.com/library/vstudio/ms173109.aspx

Value Types

## Nullable Type

Can wrap any value type to allow it to have a null value

```
Nullable<int> a = null;
int? b; // alternative syntax
int c = a.GetValueOrDefault(10);
if (b.HasValue) ...
```

Value Types

## BigInteger

- System.Numerics.BigInteger is an arbitrary-precision integer data type
  - Highly performant big integer implementation
  - Supports all the standard integer operations, including bit manipulation
  - It can be used from any .NET language, and some of the new .NET languages—such as F# and IronPython—have support built-in to the language

Selection Statements

## if

```
// block style
if (Condition1)
{
  // Condition1 is true.
}
else if (Condition2)
{
  // Condition1 is false and Condition2 is true.
}
else if (Condition3)
{
  if (Condition4)
  {
    // Condition1 and Condition2 are false. Condition3 and Condition4 are true.
  }
  else
  {
    // Condition1, Condition2, and Condition4 are false. Condition3 is true.
  }
}
else
{
  // Condition1, Condition2, and Condition3 are false.
}
```

```
// single-line style
if (Condition1)
  // Condition1 is true.
else
  // Condition1 is false.
```

Selection Statements (C# Reference)
http://msdn.microsoft.com/library/vstudio/676s4xab.aspx

## Selection Statements
## switch

1.17

```
int switchExpression = 3;
switch (switchExpression)
{
  // A switch section can have more than one case label.
  case 0:
  case 1:
    Console.WriteLine("Case 0 or 1");
    // Most switch sections contain a jump statement, such as
    // a break, goto, or return. The end of the statement list
    // must be unreachable.
    break;
  case 2:
    Console.WriteLine("Case 2");
    break;
    // The following line causes a warning.
    Console.WriteLine("Unreachable code");
  // 7 - 4 in the following line evaluates to 3.
  case 7 - 4:
    Console.WriteLine("Case 3");
    break;
  // If the value of switchExpression is not 0, 1, 2, or 3, the
  // default case is executed.
  default:
    Console.WriteLine("Default case (optional)");
    // You cannot "fall through" any switch section, including
    // the last one.
    break;
}
```

```
switch (switchExp)
{
  case 1:
    cost += 25;
    break;
  case 2:
    cost += 25;
    goto case 1;
  case 3:
    cost += 50;
    goto case 1;
  default:
    // error
    break;
}
```

## Selection Statements
## Short-Circuiting Boolean Operations

1.18

- Func2 may not be called due to short-circuiting

```
if(Func1() || Func2()) { // “or” with short-circuiting
```

```
if(Func1() && Func2()) { // “and” with short-circuiting
```

- Both functions will *always* be called

```
if(Func1() | Func2()) {
```

```
if(Func1() & Func2()) {
```

1.19

Selection Statements

## Null-Coalescing and Conditional Operations

- Null-coalescing operator (??)
  - If x is null then return -1 else return x

```
int y = x ?? -1;
```

- Conditional operator (?: ternary operator)
  - condition ? expression_if_true : expression_if_false

```
s = x != 0.0 ? Math.Sin(x)/x : 1.0;
```

  - ?: is right-associative, meaning these statements are equivalent

```
a ? b : c ? d : e
```

```
a ? b : (c ? d : e)
```

1.20

Reference Types

## What Are They?

- A reference type is a pointer to an object on the heap
- Assignment copies the memory address on the stack
  - System.String overrides this behaviour to *act* like a value type even though it is actually a reference type
  - Your types should NOT DO THIS
  - Your types should implement the ICloneable interface (and therefore provide a Clone method) instead
- Requires garbage collection to remove
  - GC does this automatically when needed
  - Your types can implement IDisposable to release memory earlier
  - See Module 11 for more details

Strings

## System.String

- *Immutable* array of char
  - New array created each time string changes so it is inefficient with many changes e.g. loop doing concatenation

```
string s1; String s2; System.String s3;
```

- Static members (also many instance members not shown here)
  - Empty: use instead of ""
  - IsNullOrEmpty(): returns true if null or ""
  - Concat(): called by operator +
  - Format(): use format patterns e.g. "{0:c}"
  - Join(): join items in string array with a separator char
  - Compare(), CompareOrdinal(): ordinal comparisons are better
  - IsInterned(): duplicate strings are pooled to save space

Strings

## System.Text.StringBuilder

- Mutable array of char
  - Much more efficient when manipulating strings
- Instance members
  - Append(), AppendLine(): add string to builder
  - AppendFormat(): add with formatting pattern
  - Capacity, EnsureCapacity(): pre-size array for more efficiency
  - MaxCapacity: Int32.MaxValue
  - Insert(): insert string into position in builder
  - Length: current size
  - Remove(), Replace()
  - ToString(): once you have finished building

Strings

## System.Security.SecureString

- Problems with System.String
  - Strings are both immutable and cannot be programmatically scheduled for garbage collection, so if a String contains sensitive information such as a password, credit card number, or personal data, there is a risk the information could be revealed after it is used
- Use SecureString when text should be kept confidential
  - Text is encrypted for privacy when being used, and deleted from computer memory when no longer needed
  - MakeReadOnly method
  - Not visible to COM

`http://msdn.microsoft.com/en-us/library/system.security.securestring.aspx`

Regular Expressions

## What Are They?

- Regular expressions can validate and process text
- When validating input, include the leading caret and trailing dollar to avoid security vulnerabilities
  - ^ means start of input; $ means end of input
  - Therefore ^\d{4}$ means only four digits
  - \d{4} means four digits, but would also match DROP table;1234

```
bool b = Regex.IsMatch("test 1234", @"\d{4}");
```

Regular Expression Library
`http://www.regexlib.com/`

The Premier website about Regular Expressions
`http://www.regular-expressions.info/`

Validating Data
`http://msdn.microsoft.com/library/vstudio/t3b36awf.aspx`

Regular Expressions

## Common Special Characters

| | | | |
|---|---|---|---|
| ^ | Start of line/string | $ | End of line/string |
| \t | Tab | \n | New line |
| \b | Boundary of word | \B | Non-boundary |
| * | Zero or more times | + | One or more times |
| ? | Zero or one time | x\|y | Either x or y |
| [xyz] | Any of the enclosed characters | [a-z] | A range of characters |
| \d \D | Digit Non-digit | \w \W | Word character non-word character |
| \s \S | White space / non-white space | \G | Match at point previous match ended |
| \040 | ASCII as octal | \u0020 | Unicode as hex |

.NET Framework Regular Expressions
`http://msdn.microsoft.com/library/hs600312.aspx`

Regular Expressions

## How to Match Using Backreferences

- Find repeating groups of characters
- Define backreference using a named group and \k
  - Named group: (?<name>chars)
- Example
  - (?<char>\w)\k<char>
  - finds adjacent paired characters

Regular Expressions

## Options

- RegExOption
  - IgnoreCase: case-insensitive matching
  - Multiline: changes meaning of ^ and $ to start and end of line, not whole string
  - Singleline: changes meaning of . to match every character (instead of every character except \n)
  - Compiled: creates an assembly; slower start-up but faster execution because the regular expression is evaluated only once
  - CultureInvariant and RightToLeft: globalization options

```
b = Regex.IsMatch("Tip", "t{1}",
  RegexOptions.IgnoreCase | RegexOptions.Compiled);
```

Regular Expressions

## How to Extract Matched Data

- Regex static methods: Match, Matches, Replace
- Match instance members: Groups, NextMatch

```
string s = Regex.Replace("test 1234", @"\d{4}", "");
```

```
MatchCollection mc = Regex.Matches("124 568", @"\d{3}");
foreach (Match m in mc)
{
  Console.WriteLine(m.Value);
}
```

Regular Expressions

## How to Replace Substrings

- Change mm/dd/yy to dd-mm-yy

```
string MDYtoDMY(string s) {
  return Regex.Replace(s, @"\b(?<month>\d{1,2})" +
    "/(?<day>\d{1,2})/(?<year>\d{2,4})\b",
    "${day}-${month}-${year}");
}
```

Encoding Text

## Encoding and Decoding

- ASCII encoding (ASCIIEncoding)
  - 7 bit bytes (0-127); inadequate for international code
- ANSI/ISO encodings (Encoding.GetEncoding method)
  - Supports code pages with language specific values (128-256)
- Unicode supports most languages
  - UTF32Encoding (32-bit integers)
  - UnicodeEncoding (16-bit integers, used internally by .NET)
  - UTF8Encoding (8-bit, 16-bit, 24-bit, 32-bit, 48-bit)
  - UTF7Encoding (7-bit ASCII, less secure and robust than UTF-8)
- Encodings are often specified in e-mails and web pages

```
<meta http-equiv="Content-Type"
  content="text/html; charset=iso-8859-1" />
```

Encoding Text

## Using the Encoding Class

- Encoding classes convert between .NET strings (UTF-16) and the specified encoding using two methods
  - GetBytes(string) returns a byte array
  - GetString(byte[]) returns a string

```
byte[] data = Encoding.UTF8.GetBytes("£23");
string s = Encoding.UTF8.GetString(data);
```

  - Hint: ToString("x2") converts a byte into hex representation
- GetEncodings returns an array of EncodingInfo

```
EncodingInfo[] eis = Encoding.GetEncodings();
foreach (EncodingInfo ei in eis)
  Console.WriteLine("{0}: {1}, {2}", ei.CodePage,
    ei.Name, ei.GetEncoding().BodyName);
```

Casting/Converting

## Converting Between Types

- Widening can be implicit; narrowing must be explicit

```
int a = 9;
long b = a;
int a = (int)b; // works with value types

Employee e = new Employee();
Person p = e;
e = (Employee)p; // could throw an exception
e = p as Employee; // reference types only
                   // and could return null
```

- If the object does not derive from the type
  - C# **as** keyword returns null
- Use **is** to check if one type derives from another

```
if (p is Person) { // safe to cast
```

Casting/Converting

## String Representations

- All types have the ToString method
  - You should override ToString in your own types to provide a string representation for an instance of your type
  - Used by debugger watch windows, when adding to List-type controls, in Write-type methods, and so on
- Many types have static Parse and TryParse methods
  - Parse method could throw an exception; TryParse returns bool
  - Allows conversion from string representation to a type instance

```
int i;
if (int.TryParse("23", out i) { // can now use i
```

```
s = "2008-03-01 10:00"; // no time zone information (see next slide)
culture = CultureInfo.CreateSpecificCulture("fr-FR");
styles = DateTimeStyles.AdjustToUniversal | DateTimeStyles.AssumeLocal;
if (DateTime.TryParse(s, culture, styles, out dateResult))
```

Casting/Converting

## Globalization Options

| DateStyles | Description |
|---|---|
| AdjustToUniversal | Parses s and, if necessary, converts it to UTC. If s includes a time zone offset, or if s contains no time zone information but styles includes the DateTimeStyles.AssumeLocal flag, the method parses the string, calls ToUniversalTime to convert the returned DateTime value to UTC, and sets the Kind property to DateTimeKind.Utc |
| AssumeLocal | Specifies that if s lacks any time zone information, it is assumed to represent a local time. Unless the DateTimeStyles.AdjustToUniversal flag is present, the Kind property of the returned DateTime value is set to DateTimeKind.Local |
| AssumeUniversal | Specifies that if s lacks any time zone information, it is assumed to represent UTC |

DateTime.TryParse Method (String, IFormatProvider, DateTimeStyles, DateTime)
`http://msdn.microsoft.com/en-us/library/9h21f14e.aspx`

Casting/Converting

## Formatting Output

### String.Format method

- Also implemented internally by Console.WriteLine and others

```
int i = 1234; string s = "Fred";
s = String.Format("{1} is {0:N0} miles away.", i, s);
```

```
Fred is 1,234 miles away.
```

String.Format Method (String, Object)
http://msdn.microsoft.com/en-us/library/fht0f5be.aspx

Composite Formatting
http://msdn.microsoft.com/en-us/library/txafckwd.aspx

Standard Numeric Format Strings
http://msdn.microsoft.com/en-us/library/dwhawy9k.aspx

Standard Date and Time Format Strings
http://msdn.microsoft.com/en-us/library/az4se3k1.aspx

Internationalization

## Formatting Data for Globalization

### CultureInfo defines

- How strings, numbers, and dates are compared
- How numbers and dates are formatted
- Which resources are retrieved during localization

### Culture can be

- Invariant: culture is not relevant
- Neutral: culture is associated with a language but not a region; en (English), fr (French), es (Spanish)
- Specific: culture is associated with a language and a region; en-US (US English), en-GB (British), fr-CA (Canadian French)

```
CultureInfo ci = new CultureInfo("fr-BE");
```

Internationalization

## Handling Dates Outside .NET

- Use ISO 8601 format code which is culture independent

```
DateTime dt = new DateTime(2008, 4, 10, 6, 30, 0);
Console.WriteLine(dt.ToString("o"));
```

```
2008-04-10T06:30:00.0000000
```

- Or use ToBinary (instance) and FromBinary (static)
  - 64-bit value encodes Kind and Ticks
  - Includes local time zone and automatically adjusts

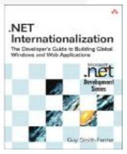
```
// executes on a machine in London
DateTime dtLocalLondon = DateTime.Now;
long b = dtLocalLondon.ToBinary();
```

```
// running on a machine in Paris
DateTime dtLocalParis = DateTime.FromBinary(b);
```

Internationalization

## Threads and Culture

- Threads have two culture properties
  - CurrentCulture: globalizes code
    Automatically set from regional setting in OS (en-GB)
    Should always be a specific culture
  - CurrentUICulture: localizes user interface
    Automatically set from version of OS (en-US)
    Can be a neutral culture
  - Can be replaced with a new CultureInfo instance
- Specific culture's Parent is neutral, then Invariant
- .NET Internationalization: The Developer's Guide to Building Global Windows and Web Applications
  - Guy Smith-Ferrier

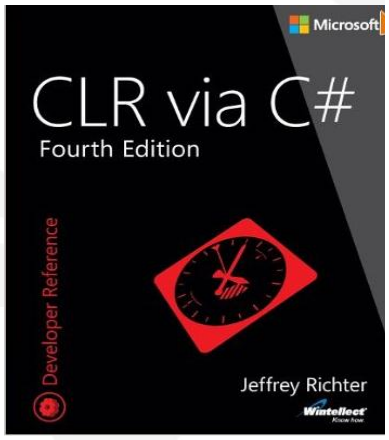


Miscellaneous
## Obsolete Types and Members

- Some types and members are now considered to be obsolete (deprecated)
  - Check the MSDN documentation
  - The compilers will also warn you
- Examples
  - XmlValidatingReader class
  - EventLog.CreateEventSource method
- Apply ObsoleteAttribute to your own types and members

Miscellaneous
## Further Study

- BCL Team Blog
  - Great for inside information about how and why the BCL works
  - http://blogs.msdn.com/b/bclteam/
- CLR via C#, 4th Edition
  - Jeffery Richter
  - Dig deep and master the intricacies of the common
    language runtime (CLR) and the .NET Framework



# Module 2
Creating Methods, Handling Exceptions, and Monitoring Applications

Creating Methods, Handling Exceptions, and Monitoring Applications

## Contents

| Topic | Slide |
|---|---|
| Methods | 3 |
| Extension Methods | 5 |
| Exceptions | 6 |
| Diagnostics | 7 |
| Managing Code | 16 |

**Exam Topic: Implement exception handling**
- ❑ Handle exception types (SQL exceptions, network exceptions, communication exceptions, network timeout exceptions) (2-12)
- ❑ Catch typed vs. base exceptions (2-13)
- ❑ Implement try-catch-finally blocks (2-14)
- ❑ Throw exceptions (2-16)
- ❑ Determine when to rethrow vs. throw (2-16)
- ❑ Create custom exceptions (5-12)

**Exam Topic: Debug an application**
- ❑ Create and manage compiler directives (14)
- ❑ Choose an appropriate build type (2-18)
- ❑ Manage programming database files and symbols

**Exam Topic: Create types**
- ❑ Create methods, extension methods, optional and named parameters, and indexed properties (2-3)
- ❑ Create overloaded and overridden methods (2-8)

**Exam Topic: Implement diagnostics in an application**
- ❑ Implement logging and tracing (2-17)
- ❑ Profiling applications (2-19)
- ❑ Create and monitor performance counters (2-20)
- ❑ Write to the event log (2-18)

2.3

Methods

## Overloading Methods and Constructors

- Can have multiple implementations as long as the <u>input</u> parameters are different data types

```
double Calc() {
  return ...
}
double Calc(string s) {
  return ...
}
int Calc(string s) { // compile error
  return ...
}
```

2.4

Methods

## Optional and Named Parameters

- Optional parameters

```
public StreamReader OpenTextFile(
  string path, Encoding encoding = null,
  bool detectEncoding = true, int bufferSize = 1024);
```

- Named arguments must be last

```
OpenTextFile("foo.txt", Encoding.UTF8,
  bufferSize: 4096);
```

- Named arguments can be in any order

```
OpenTextFile(
    bufferSize: 4096,
    path: "foo.txt",
    detectEncoding: false);
```

  - Non-optional arguments must be specified

Extension Methods
## How to Create Extension Methods

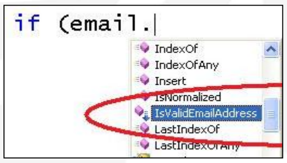
2.5

- Extension methods allow you to add methods to a type without inheritance or recompilation
  - Create a static class with a static method that uses the *this* keyword before a parameter to specify the type to extend

```
namespace MyExtensions {
  public static class StringExtensions {
    public static bool IsValidEmailAddress(this string s) {
```

  - Import the namespace containing the class

```
using MyExtensions;
```



```
if (email.|
```

Exceptions
## How to Throw, Catch, and Clean Up

2.6

- Thrown when an exceptional action occurs

```
throw new ArgumentException("message");
```

- System.Exception is base class for all exceptions
  - Includes StackTrace with line numbers
  - Deriving from ApplicationException no longer recommended
- Try...Catch
  - Multiple catch blocks are allowed from most general to most specific type of exception

```
catch ... {
  // rethrow with original stack trace
  throw;
```

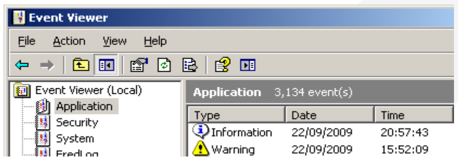
- Try...Catch...Finally
  - Finally block executes before control passed up call stack
  - Often used to release unmanaged resources

2.7

Diagnostics

## Logging to Event Viewer

- Event Viewer is used by admins to view event logs
  - System: non-security OS events
  - Security: auditing events; applications cannot write to this log
  - Application: for applications that do not create their own log; can be filtered by Event Source (typically the application that wrote the event entry)
  - Custom logs



| Type | Date | Time |
|---|---|---|
| Information | 22/09/2009 | 20:57:43 |
| Warning | 22/09/2009 | 15:52:09 |

- Only first eight characters are significant in log names!

2.8

Diagnostics

## Debugger...Attributes

- Use to customize how a type instance appears in watch

```
[DebuggerDisplay("First Name: {FirstName}")]
public class Person
```

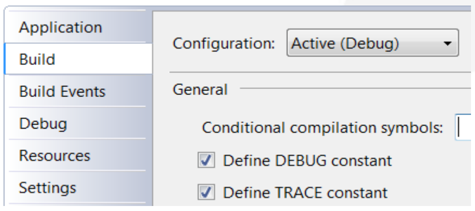
| | |
|---|---|
| DebuggerBrowsable | Should this field display in watch windows?<br>Values: Never, Collapsed, RootHidden |
| DebuggerDisplay | How should this type display in watch windows?<br>Use custom string to format field values |
| DebuggerHidden | Prevent breakpoints being set inside the method this is applied to |
| DebuggerStepThrough | Apply to method to step over the code when debugging (but it still executes the method) |
| DebuggerTypeProxy | Override how a given type is shown and then specify how you want it shown |
| DebuggerVisualizer | Specifies which debug visualizer to use for this code |

Diagnostics

## Logging Debugging and Trace Information

2.9

- Debug and Trace objects can be used to capture information
  - Only available when DEBUG and TRACE compiler constants are defined
- In the default Solution Configurations
  - Both DEBUG and TRACE are set for Debug configuration
  - Only TRACE is set for Release configuration



Configuration: Active (Debug)

Conditional compilation symbols:

☑ Define DEBUG constant

☑ Define TRACE constant

Diagnostics

## Debug and Trace methods and properties

2.10

| | |
|---|---|
| Assert | Evaluates a condition, breaks and displays a message box if the condition evaluates to false; unless .config has: <assert assertuienabled="false" logfilename="..."/> |
| Fail | Outputs a failure message box including stack trace |
| Write, WriteIf | Write to the listeners without a line break |
| WriteLine, WriteLineIf | Write to the listeners with a line break |
| Print | Same as WriteLine; for compatibility with VB6 |
| Indent, Unindent | Indents the output; IndentLevel shows current level, IndentSize controls amount of indentation |
| Flush | Calls Flush on attached listeners; or set AutoFlush |
| Close | Calls Close on attached listeners |
| Listeners | Collection of listener objects; default is Output window which does not need to be Flushed or Closed |

Diagnostics

## Using Listeners

2.11

- Debug and Trace share the same Listeners collection so adding a listener to one makes it available to both
- Listeners inherit from TraceListener and determine where the output from is written to
  - DefaultTraceListener: Output window of VS; named "Default"
  - ConsoleTraceListener: a Console window
  - TextWriterTraceListener: a text file
  - DelimitedListTraceListener: a delimited text file
  - XmlWriterTraceListener: an XML file
  - EventSchemaListener: an XML file that conforms to a schema
  - EventLogTraceListener: an event log
  - WebPageTraceListener: integrate with ASP.NET Trace

Diagnostics

## Configuring Tracing Using a .config File

2.12

- Adds a text file listener and removes the default

```
<system.diagnostics>
  <trace autoflush="true">
    <listeners>
      <add name="configText"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="output.txt" />
      <remove name="Default" />
    </listeners>
  </trace>
</system.diagnostics>
```

- Writing to the Application event log

```
<add name="configEventLog"
  type="System.Diagnostics.EventLogTraceListener"
  initializeData="Application" />
```

Diagnostics

## Shared Listeners

- Can set up shared listeners for use by Trace and one or more TraceSources

```
<trace> <!-- Trace -->
  <listeners>
    <add name="sharedLogger" />
  </listeners>
</trace>
<sources>
  <source name="ts"> <!-- new TraceSource("ts") -->
    <listeners>
      <add name="sharedLogger" />
    </listeners>
  </source>
</sources>
<sharedListeners>
  <add name="sharedLogger"
    type="System.Diagnostics.ConsoleTraceListener" />
</sharedListeners>
```

Diagnostics

## Conditional Directive

- When the C# compiler encounters an #if directive, followed eventually by an #endif directive, it will compile the code between the directives only if the specified symbol is defined

```
#if DEBUG
    Console.WriteLine("Debug version");
#endif
```

- Either select the Debug check box in Project properties or add following line of code or use /debug option

```
#define DEBUG
```

#if (C# Reference)
http://msdn.microsoft.com/en-us/library/4y6tbswk.aspx

Diagnostics

## Conditional Attribute

2.15

- Indicates to compilers that a method call or attribute should be ignored unless a specified conditional compilation symbol is defined

```
[Conditional("CONDITION1")]
public static void Method1(int x)
{
  Console.WriteLine("CONDITION1 is defined");
}
```

ConditionalAttribute Class
http://msdn.microsoft.com/en-us/library/system.diagnostics.conditionalattribute.aspx

Managing Code

## Defining Regions

2.16

- #region lets you specify a block of code that you can expand or collapse when using the outlining feature of the Visual Studio Code Editor
  - In longer code files, it is convenient to be able to collapse or hide one or more regions so that you can focus on the part of the file that you are currently working on

```
#region MyClass definition
public class MyClass
{
    static void Main()
    {
    }
}
#endregion
```

#region (C# Reference)
http://msdn.microsoft.com/en-us/library/9a1ybwek(v=vs.110).aspx

# Module 3
Developing the Code for a Graphical Application

## Developing the Code for a Graphical Application
## Contents

**Exam Topic: Create and implement events and callbacks**
- ❑ Create event handlers (3-17)
- ❑ Subscribe to and unsubscribe from events (3-19)
- ❑ Use built-in delegate types to create events (21)
- ❑ Create delegates (19)
- ❑ Lambda expressions (25)
- ❑ Anonymous methods (20)

**Exam Topic: Query and manipulate data and objects by using LINQ**
- ❑ Query data by using operators (projection, join, group, take, skip, aggregate) (35, 47, 53)
- ❑ Create method-based LINQ queries (35)
- ❑ Query data by using query comprehension syntax (3-15)
- ❑ Select data by using anonymous types (49)
- ❑ Force execution of a query (39, 46)

**Exam Topic: Store data in and retrieve data from collections**
- ❑ Store and retrieve data by using dictionaries, arrays, lists, sets, and queues (3-10)
- ❑ Initialize a collection (3-13)
- ❑ Add and remove items from a collection (3-13)

Value Types (struct)

## How to Create User-Defined Value Types

- Structures define value types
  - Can have: constructors, fields, methods, operators

```
struct Cycle {
  // ...
}
```

- Benefits
  - If < 16 bytes of field data, more efficient than classes
  - Point has X and Y (Int32), therefore 8 bytes of field data
- Limitations
  - Cannot inherit from structures

Enumerations (enum)

## What Are They?

- List of constants derived from integer types
  - If not specified, default is System.Int32 (int, Integer)
- Useful for simple lookups

```
enum Fruit : byte {
  Apple = 1,
  Banana,
  Cherry
}
```

```
Fruit f = Fruit.Cherry;
f = (Fruit)2;
f = Enum.Parse(typeof(Fruit),
  "apple", true) as Fruit;
```

## Enumerations (enum)
## Bitwise Operations

- What are the values of a, b, c, d?

```
a = 2 & 8;
b = 2 | 8;
c = 8 & 9;
d = 8 | 9;
```

- Bitwise operations apply to the *bits*

```
2 is 0010
8 is 1000
9 is 1001
```

- So the values are…

```
a = 0  = 0000
b = 10 = 1010
c = 8  = 1000
d = 9  = 1001
```

## Enumerations (enum)
## Bitwise Operations and FlagsAttribute

- Apply [Flags] to enumeration to allow recognition that the values could be combined in bitwise operations
- You must manually set values to 1, 2, 4, 8, and so on

```
[Flags] enum Fruit {
  None = 0, Apple = 1, Banana = 2,
  Cherry = 4, Date = 8, Elderberry = 16
}
```

- Strings will now output correctly

```
Fruit f = (Fruit)3;
Console.WriteLine(f);
```

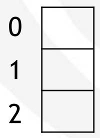
```
Apple, Banana
```

- Without Flags it would print as 3

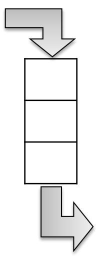
3.7

Collections
## What Are They?

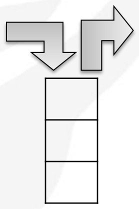
- Resizable data structures that store multiple objects
- List collections implement IList and are indexed
  - ArrayList: object
  - StringCollection: string
  - List<T>: strongly-typed



- Sequential collections have custom add/remove methods
  - Queue: object
  - Queue<T>: strongly-typed



  - Stack: object
  - Stack<T>: strongly-typed



3.8

Collections
## ArrayList and List<T>

```
ArrayList al = new ArrayList(); // contains System.Object
al.Capacity = 5; // pre-size the collection
al.Add(123);
string[] words = { "more", "or", "less" };
al.AddRange(words); // insert items as separate objects
al.Insert(3, "Hey Dude!"); // insert into a position in list
al[3] = "Hey Buddy!"; // change the value
```

```
List<int> il = new List<int>();
il.Capacity = 5; // pre-size the collection
il.Add(123);
int[] numbers = { 27, 36, 95 };
il.AddRange(numbers); // insert items as separate objects
il.Insert(3, 56); // insert into a position in list
il[3] = 57; // change the value
```

3.9

Collections

## Iterating Items

- Compiler converts For Each statements into calls to these interfaces
  - IEnumerable, IEnumerable<T>: GetEnumerator()
  - IEnumerator, IEnumerator<T>: Reset(), MoveNext(), Current

```
foreach (object o in c) {
  Console.WriteLine(o);
}
```

```
IEnumerator i =
  c.GetEnumerator();
while (i.MoveNext())
  Console.WriteLine(i.Current);
```

3.10

Collections

## Common Interfaces for List Collections

- ICollection and ICollection<T>
  - Count, CopyTo
  - IsSynchronized: is type thread-safe?
  - SyncRoot: returns object to be used for thread synchronization
- IList and IList(T) are for indexed collections
  - IsFixedSize, IsReadOnly
  - Add(object), Insert(object, index)
  - Contains(object), IndexOf(object)
  - Remove(object), RemoveAt(index), Clear()
  - Sort(), Sort(IComparer), Sort(Comparison<T>), Sort(index, count, IComparer)

Collections

## Sorting Collections with IComparable

- To be able to sort collections the type in the collection can implement IComparable or IComparable<T>
  - CompareTo returns -1 if less than, 0 if equal, 1 if greater than

```
public class Person : IComparable<Person> {
  public string FirstName { get; set; }
  ...
  public int CompareTo(Person other) {
    return this.FirstName.CompareTo(other.FirstName);
```

- Array.Sort, List<T>.Sort

```
Person[] people = ... ;
Array.Sort(people);
```

```
List<Person> morePeople = ... ;
morePeople.Sort();
```

- If you have not implemented IComparable it throws
  - InvalidOperationException, “Failed to compare two elements”

Collections

## Sorting with IComparer<T> and Comparison<T>

- If your type does not (or cannot) implement IComparable then create a new class that implements IComparer or IComparer<T>

```
public class PersonComparer : IComparer<Person> {
  public int Compare(Person p, Person other) {
    return p.LastName.CompareTo(other.LastName);
```

```
List<Person> morePeople = ... ;
morePeople.Sort(new PersonComparer());
```

- Or use Comparison with a lambda expression

```
morePeople.Sort(new Comparison<Person>(
  (p, other) =>
    p.LastName.CompareTo(other.LastName)
  ));
```

Collections
## Building Collections
3.13

- Easily build a "collection" by using yield keyword
  - Signals to the compiler that the method is an iterator block
  - Used together with the return or break keywords

```
public IEnumerable Power(int number, int exponent) {
  int counter = 0;
  int result = 1;
  while (counter++ < exponent) {
    result = result * number;
    // returns result, pauses until the next iteration
    yield return result;
  }
}
```

```
foreach (int i in Power(2, 8)) {
  Console.Write("{0} ", i);
}
```

```
// 2 4 8 16 32 64 128 256
```

Collections
## Dictionaries
3.14

- Collection of key/value pairs
  - Key must be unique (see next slide)
  - Good for fast lookups based on key
  - Not usually sorted or indexed

| Key | Value |
|---|---|
| | |
| | |
| | |

- Common interfaces
  - IDictionary and IDictionary<TKey, TValue>
- Choose based on size of collection
  - Hashtable: >10
  - ListDictionary: <10
  - HybridDictionary: switches between Hashtable & ListDictionary
  - Dictionary<TKey, TValue>: generic dictionary is good for all sizes

Collections

## Understanding Equality

3.15

- Every type derives from System.Object
  - GetHashCode(): you should override this to return a unique integer; the base implementation partially uses memory address in an attempt to generate a unique integer
  - Equals(object): return true/false
- How dictionaries check for duplicate keys
  - Call GetHashCode() on both keys and compare the integers
  - If both have same hash then Equals() called
- IEqualityComparer and IEqualityComparer<T> interfaces
  - GetHashCode and Equals methods
  - Implement to use a custom mechanism to check for duplicates
  - Pass instance into constructor of Hashtable or Dictionary

Collections

## Adding, Modifying, and Iterating Dictionaries

3.16

- Add method throws exception if key already exists

```
Hashtable ht = new Hashtable();
ht.Add("key1", "value1");
ht.Add("key1", "value2"); // exception thrown
```

- Set item either adds or modifies if key already exists

```
ht["key2"] = "value3"; // adds
ht["key2"] = "value4"; // modifies
```

- IDictionaryEnumerator
  - Each item is a DictionaryEntry or KeyValuePair<TKey, TValue>

```
foreach (DictionaryEntry entry in emails) {
  // entry.Key (object), entry.Value (object)
}
```

Collections

## SortedSet<T>

- SortedSet<T> collection along with an ISet<T> interface
- SortedSet<T> uses a self-balancing tree which maintains data in sorted order for performance guarantees with insertion, deletion, and searches
- Both the new SortedSet<T> and the existing HashSet<T> implement ISet<T>

Collections

## Tuples

- A tuple is a simple generic data structure that holds an ordered set of items of heterogeneous types
- We are providing common tuple types in the BCL to facilitate language interoperability and to reduce duplication in the framework
- Tuples are supported natively in languages such as F# and IronPython, but are also easy to use from any .NET language such as C# and VB

Delegates and Events

## What Is A Delegate?

- Type-safe function pointer
  - Delegate must match signature of the method you want to call

```
// method I want to call
int M1(string s) {
  return s.Length;
}
```

```
delegate int Del(string s);
```

```
Del d = new Del(M1);
int i = d("Fred");
// d.Invoke("Fred")
```

Delegates and Events

## Why are Delegates Useful?

- Treat methods as data
  - For example, create a queue of methods to call
- Anonymous delegates
  - Simplify code by removing need for defining a private method

```
Button1.Click += delegate { Debug.Write("Clicked"); };
```

- Can be invoked asynchronously using BeginInvoke
- Lambda expressions (used in LINQ)
  - Lambda expressions can be used in place of a delegate instance
- Loose-binding of types; cleaner type design
  - Foundation of events

Delegates and Events

## What Is an Event?

- Events are built on delegates
- EventHandler is a pre-defined delegate that conforms to convention of method signature for event handlers
  - sender (System.Object), e (System.EventArgs or derived class)
  - EventHandler<T> is the generic version
- To declare an event

```
public event EventHandler<LightEventArgs> Socket;
```

  - Use of event keyword when declaring delegate only allows += and -= operators, not =
- To raise an event:

```
if(Socket != null) Socket(this, new LightEventArgs());
```

Language Features

## Object Initializers

- C# 2.0 and earlier

```
Person person = new Person();
person.FirstName = "John";
person.LastName = "Smith";
person.Age = 32;
```

- C# 3.0 and later

```
Person person = new Person()
  { FirstName="John", LastName="Smith", Age=32 };
```

Language Features

## Array and Collection Initializers

- Initialize an array of a simple type

```
string[] names = new string[]
  { "Scott", "Bill", "Susanne" };
```

- Initialize a collection of a complex type

```
List<Person> people = new List<Person>() {
  new Person()
    { FirstName = "Scott", LastName = "Smith", Age = 32 },
  new Person()
    { FirstName = "Bill", LastName = "Gates", Age = 50 },
  new Person()
    { FirstName = "Susanne", LastName = "Smith", Age = 32 }
};
```

  - Types must implement IEnumerable and have suitable Add method

Language Features

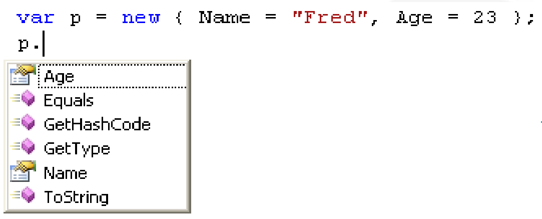
## Inferred and Anonymous Types

- Infers type of local variables *at compile time*

```
var name = "Mark";
```

  - Compiler must be able to infer the type so you must assign an initial value, which can be returned from a method call

- Anonymous types can be inferred from an object initializer statement



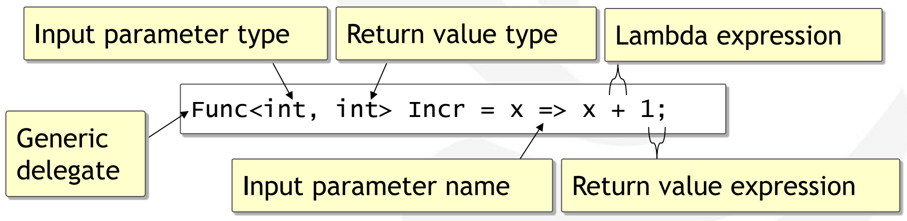
```
var p = new { Name = "Fred", Age = 23 };
p.
```

  - Instances of anonymous types are immutable in C#

Lambda Expressions

## What Are They?

- A lambda expression is simply a *nameless* function
  - Can be used wherever a delegate is valid



```
Func<int, int> Incr = x => x + 1;
```

- Note: Func is a generic delegate defined by Microsoft

Lambda Expressions

## Syntax

- A lambda expression syntax

```
(input parameters) => expression
```

```
input parameter => expression
```

- Example, inferring input types from delegate

```
MyDelegate d = (x, y) => x == y;
```

- Example, explicitly defining input types

```
MyDelegate d = (int x, string y) => x > y.Length;
```

- Must use parentheses with zero input parameters; parentheses are only optional with one parameter

```
MyDelegate d = () => SomeMethod();
```

Lambda Expressions

## Generic Delegates Often Used With Lambdas

- Func(TResult)
  - For lambda expressions with no inputs
- Func(T, TResult)
  - For lambda expressions with one input parameter

```
Func<int, bool> myFunc = x => x == 5;
bool result1 = myFunc(4); // returns false
bool result2 = myFunc(5); // returns true
```

- Func(T1, T2, TResult)
  - For lambda expressions with two input parameters
- Func(T1, T2, T3, TResult) and so on
- Predicate(T): one input and always returns a Boolean

Lambda Expressions

## Lambda Statements and Multi-Line Expressions

- Lambda statements are nameless methods that return void
- Statement lambda syntax

```
(input params) => { statements; }
```

- Statement lambdas cannot be used in expression trees and therefore *cannot be used in LINQ queries*
- Generic delegates for use with lambda statements
  - Action(T), Action(T1, T2), and so on
- Lambda expressions can also have multiple statements, but must return a value

```
(input parameters) => { statements; return value; }
```

LINQ

## What Is It?

3.29

- Most databases understand SQL...
  - ...but to C# 2.0 and VB 8.0, an SQL statement is just a string
  - LINQ integrates query syntax to a .NET language
- LINQ is made up of three parts
  - Providers for data sources (required)
    - LINQ to Objects, LINQ to SQL, LINQ to Entities, LINQ to XML, LINQ to SharePoint, LINQ to Amazon, and so on
  - Extensions to the base class libraries (required)
    - System.Linq.Enumerable and System.Linq.Queryable classes in System.Core.dll assembly
  - Extensions to the languages and compilers (optional)
    - C# keywords: from, select, orderby, and so on
    - VB keywords: From, Select, Order By, and so on

LINQ

## Provider Limitations

3.30

- Theoretically, once you learn LINQ, you can query any LINQ provider...
  - ...but some LINQ providers have limitations
- For example, LINQ to SQL
  - This LINQ provider must eventually convert the expression tree created by your LINQ statements into Transact-SQL statements, so not all LINQ statements are fully supported, or might be implemented in ways that you do not like
- LINQ to Objects supports all features, but other providers may lack support for some features, or display unexpected behaviour
  - For example, T-SQL cannot order inner queries
  - So use ToList(T) method to get data from LINQ to SQL, and then use LINQ to Objects on the result

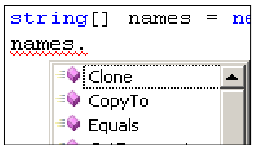
3.31

LINQ

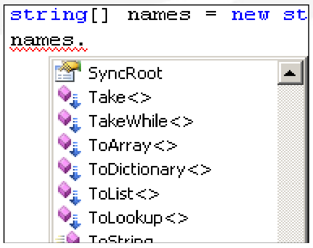
## Enumerable and Queryable classes (System.Linq)

- LINQ requires types to implement interfaces to support it’s features: IEnumerable(T) or IQueryable(T)
  - If a type does not, IEnumerable has some extension methods: OfType(T), Cast(T) that can convert to the generic versions
- LINQ uses extension methods defined by Enumerable and Queryable classes in the System.Linq namespace
  - Importing the namespace allows the extension methods to be used on any type that implements IEnumerable(T) or IQueryable(T)

```
using System.Linq;
```



```
string[] names = ne
names.
```



```
string[] names = new st
names.
```

3.32

LINQ

## IEnumerable and IQueryable interfaces

- IEnumerable means LINQ to Objects
  - All data must be materialized locally before extension methods are applied
- IQueryable means LINQ to Entities (or LINQ to SQL)
  - Data retrieval is deferred
  - An expression tree is created and only when the query is enumerated (with foreach) or one of the ToXxx methods is called will the expression tree be converted into an SQL statement and executed to retrieve the appropriate data
  - Use ToString (with DbContext) or ToTraceString (with ObjectContext) to see the T-SQL that will be executed
  - Use a To*Something* method to retrieve data, then you can use LINQ to Objects without limitations

LINQ

## To create extension methods for LINQ

- Create a static class that extends IEnumerable<T>
  - Returns a scalar

```
namespace System.Linq {
  public static class MyLinqExtensions {
    public static double MyAggregate<T>(
      this IEnumerable<T> input) {
```

  - Returns a sequence

```
    public static IEnumerable<T> MyProcessor<T>(
      this IEnumerable<T> input) {
```

LINQ

## Language-Level Support (Syntactic Sugar)

- Instead of using extension methods and lambda expressions, C# and Visual Basic provide simplified syntax for queries

```
var query =
  from name in names
  where name.StartsWith("A")
  orderby name
  select name;
```

- Equivalent to...

Lambda expression

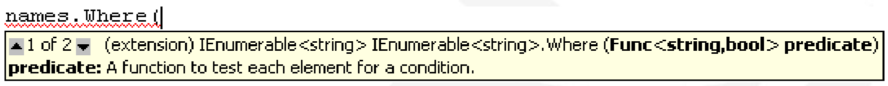
```
var query =
  names.Where(name => name.StartsWith("A"))
  .OrderBy(name => name);
```

```
// optional
.Select(name => name)
```

LINQ Extension Methods

## Where

- Where takes a Func<TInput,TReturn> generic delegate as input (so we can use a lambda expression instead)
  - Expression must have a single input parameter (of whatever type T in IEnumerable(T) is) and must return a boolean
  - So for the names array of strings



```
names.Where(
```

▲1 of 2 ▼ (extension) IEnumerable<string> IEnumerable<string>.Where (**Func<string,bool> predicate**)
**predicate:** A function to test each element for a condition.

```
string[] names = ...;
... names.Where(name => name.StartsWith("A"));
```

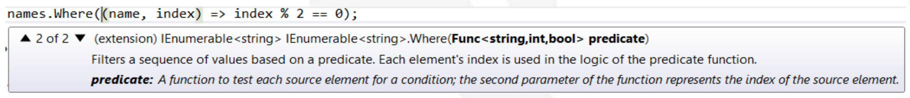
  - For an array of Person

```
Person[] people = ...;
... people.Where(p => p.Age > 18);
```

LINQ Extension Methods

## Where and Select Method Index Values

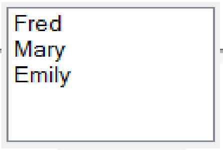
- Where (and Select) take Func<TInput,int,TReturn> delegate as input so you can filter based on index



```
names.Where((name, index) => index % 2 == 0);
```

▲ 2 of 2 ▼ (extension) IEnumerable<string> IEnumerable<string>.Where(**Func<string,int,bool> predicate**)
Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.
***predicate:*** *A function to test each source element for a condition; the second parameter of the function represents the index of the source element.*

```
string[] names = new string[] { "Fred", "George",
  "Mary", "Sally", "Emily", "Harry" };
var query = names.Where((name, index) => index % 2 == 0);
var results = query.ToArray();
listBox1.Items.AddRange(results);
```

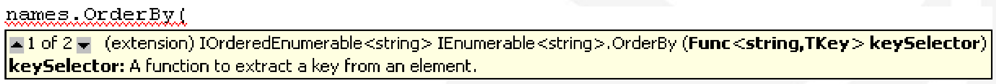


```
Fred
Mary
Emily
```

LINQ Extension Methods

## OrderBy

- OrderBy takes a Func<TInput,TKeySelector> generic delegate as input
  - Lambda expression must have a single input parameter (of whatever type T in IEnumerable(T) is) and can return any type
  - For names string array, we might want to order by the number of characters in each string entry



```
names.OrderBy(
```

1 of 2 (extension) IOrderedEnumerable<string> IEnumerable<string>.OrderBy (**Func<string,TKey> keySelector**)
**keySelector:** A function to extract a key from an element.

```
... names.OrderBy(name => name.Length);
```

  - For names string array, we might want to order by the entire entry (which looks strange but is necessary due to the syntax!)

```
... names.OrderBy(name => name);
```

LINQ Extension Methods

## OrderBy and GroupBy

- Usually you want to order first, then group

```
string[] names = new string[]
  { "Fred", "George", "Gary", "Emily" };
var query = names.OrderBy(name => name)
  .GroupBy(name => name[0]);
```

```
E
  Emily
F
  Fred
G
  Gary
  George
```

- If you group first, you need to specify how to order the *group*, not the original items

```
string[] names = new string[]
  { "Fred", "George", "Gary", "Emily" };
var query = names.GroupBy(name => name[0])
  .OrderBy(group => group.Key);
```

```
E
  Emily
F
  Fred
G
  George
  Gary
```

LINQ Extension Methods

## Chaining, Deferred Execution, and Materialization

- Most extension methods return IEnumerable(T) so that they can be chained

```
... names.Where(name => name.StartsWith("A"))
         .OrderBy(name => name);
```

  - The extension methods will be processed in order
  - Methods that return a sequence of values do not consume the target data until the query is enumerated (deferred execution)
  - The query is not executed until enumerated over and will re-execute over the original data each time so will detect changes
  - Materialize a copy with ToArray, ToList, and so on
  - Methods that return a singleton value execute and consume the target data immediately
  - Do not assume the entries in a sequence are ordered unless you explicitly specify

LINQ Extension Methods

## Enumerable Static Methods

- Empty<T>

```
IEnumerable<Person> empty = Enumerable.Empty<Person>();
```

- Range

```
IEnumerable<int> squares =
  Enumerable.Range(4, 3).Select(x => x * x);
```

```
/* 16
   25
   36 */
```

- Repeat

```
IEnumerable<string> madness = Enumerable.Repeat(
  "All work and no play makes Jack a dull boy.", 20);
```

```
/* All work and no play makes Jack a dull boy.
   All work and no play makes Jack a dull boy.
   All work and no play makes Jack a dull boy.
```

LINQ Extension Methods

## Non-Deferred, Scalar Return Value Methods

### Aggregate()

- Creates accumulations over a sequence of elements with a lambda expression, or use one of the built-in aggregates:
  - Average(), Count(), LongCount(), Max(), Min(), Sum()

```
var query = db.Products;
decimal minPrice = query.Min(p => p.ListPrice);
```

### All(), Any()

- Returns true if all or any elements satisfy the lambda expression

```
if(names.Any(name => name.StartsWith("A")))
```

### SequenceEqual()

- Returns true if the two sequences contain the same elements in the same order

LINQ Extension Methods

## Non-Deferred, Single Item Return Value Methods

### First, FirstOrDefault, Last, LastOrDefault, ElementAt, ElementAtOrDefault, DefaultIfEmpty(def)

- First, last or at index, or default if sequence is empty
- N.B. Default for type, e.g. default(int) would be 0

```
Person p = people.First(); // might throw exception
```

```
Person p = people.FirstOrDefault(); // might return null
```

### Single, SingleOrDefault

- Returns a specific member of a sequence, or default value, or throws exception if more than one item in sequence

```
Person q = people.Where(p => p.ID == 123).Single();
```

```
Person q = people.Single(p => p.ID == 123);
```

LINQ Extension Methods

## Deferred, Multiple Item Return Value Methods

- Where
  - Filters the sequence by specific criteria
- IEnumerable: OrderBy, OrderByDescending, Reverse
  IOrderedEnumerable: ThenBy, ThenByDescending
  - Ascending and descending chained sorts, or reverse the order
- Skip, SkipWhile
  - Skips n members, or while lambda expression returns true
- Take, TakeWhile
  - Takes n members, or while lambda expression returns true
- Distinct, Except, Intersect, Concat, Union, Zip
  - Sequence where members are distinct, differ, match, all, or zipped 1-1, 2-2, 3-3, and so on

LINQ Extension Methods

## Comparing Concat and Union

- Two sequences of integers

```
int[] lastYearScores = { 88, 56, 23, 99, 65 };
int[] thisYearScores = { 93, 78, 23, 99, 90 };
```

- Concat (includes duplicates)

```
foreach (var item in
  lastYearScores.Concat(
    thisYearScores))
```

```
88, 56, 23, 99, 65,
93, 78, 23, 99, 90
```

- Union (removes duplicates)

```
foreach (var item in
  lastYearScores.Union(
    thisYearScores))
```

```
88, 56, 23, 99, 65,
93, 78, 90
```

## LINQ Extension Methods
## As*Something* Conversions

- AsEnumerable<T>()
  - Convert IEnumerable to IEnumerable<T>
  - “Execute” a query without creating a local collection
- AsQueryable<T>() Convert IQueryable to IQueryable<T>
- AsParallel()
  - PLINQ is designed to exploit opportunities for parallelization, however, not all queries benefit
  - It partitions the data source into segments, and then executes the query on separate threads on multiple processors
  - The overhead can be more expensive than the speedup so PLINQ may decide to execute some or all of the query sequentially

```
from cust in customers.AsParallel().WithExecutionMode(
  ParallelExecutionMode.ForceParallelism)
```

## LINQ Extension Methods
## To*Something* Conversions

- ToList and ToArray: return flat collection of results
- ToDictionary
  - One-to-one mapping of keys to objects
  - Requires a lambda to define property to use for the key

```
Dictionary<string, Product> products =
  db.Products.ToDictionary(p => p.ProductName);
```

- ToLookup
  - One-to-many mapping of keys to collections
  - Requires a lambda to define property to use for the key

```
ILookup<string, Product> products =
  db.Products.ToLookup(p => p.Category.Name);
IEnumerable<Product> bikes = products["Bike"];
```

## Projection
## Primitive Results

- p is a Product

```
var query =
  from p in db.Products
  select p;
```

```
var query = db.Products
  .Select(p => p); //optional
```

```
List<Product> results = query.ToList();
```

- p.Name is a string

```
var query = from p in db.Products
            select p.Name;

List<string> results = query.ToList();
```

## Projection
## Projecting into Types

- A type that defines a subset of product information

```
public class ProductInfo {
  public string Name;
  public decimal Price; }
```

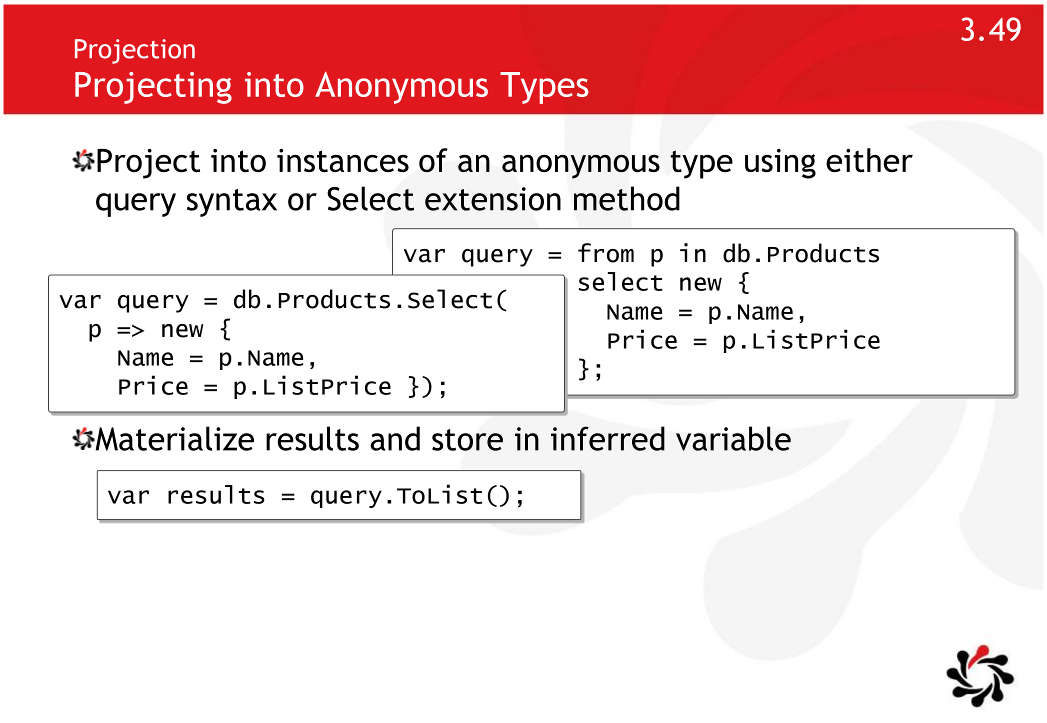
- Project into instances of this type using either query syntax or Select extension method

```
var query = from p in db.Products
            select new ProductInfo() {
              Name = p.Name,
              Price = p.ListPrice
            };
```

```
var query = db.Products.Select(
  p => new ProductInfo() {
    Name = p.Name,
    Price = p.ListPrice });
```

- Materialize results

```
List<ProductInfo> results = query.ToList();
```



3.49

Projection

## Projecting into Anonymous Types

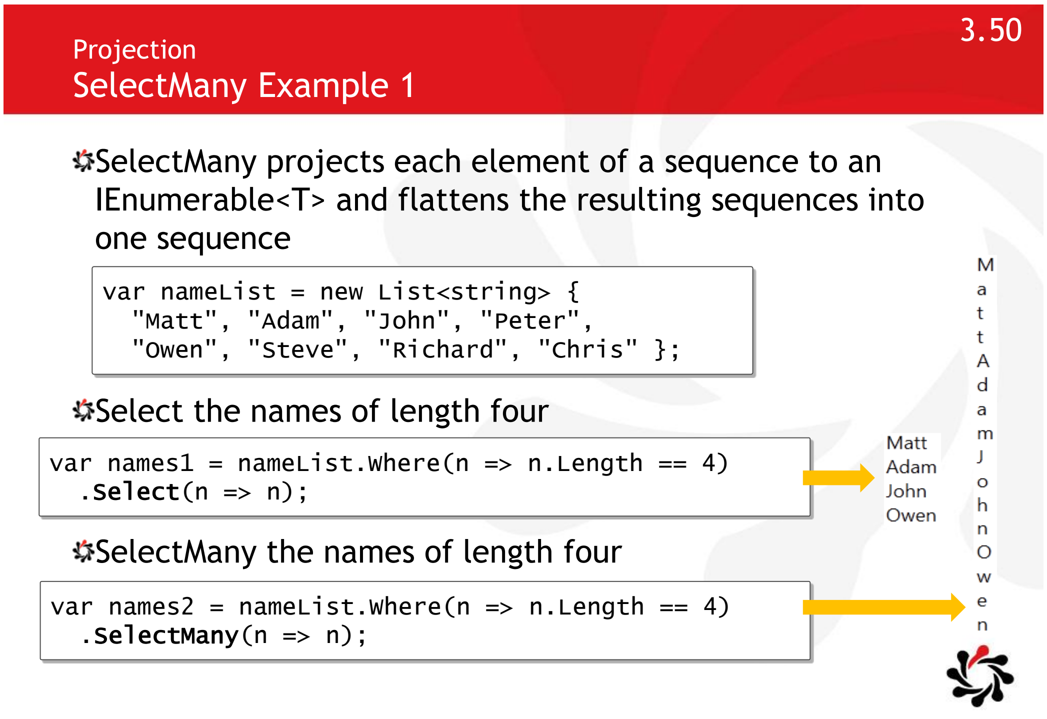
- Project into instances of an anonymous type using either query syntax or Select extension method

```
var query = db.Products.Select(
  p => new {
    Name = p.Name,
    Price = p.ListPrice });
```

```
var query = from p in db.Products
            select new {
              Name = p.Name,
              Price = p.ListPrice
            };
```

- Materialize results and store in inferred variable

```
var results = query.ToList();
```



3.50

Projection

## SelectMany Example 1

- SelectMany projects each element of a sequence to an IEnumerable<T> and flattens the resulting sequences into one sequence

```
var nameList = new List<string> {
  "Matt", "Adam", "John", "Peter",
  "Owen", "Steve", "Richard", "Chris" };
```

- Select the names of length four

```
var names1 = nameList.Where(n => n.Length == 4)
  .Select(n => n);
```

```
Matt
Adam
John
Owen
```

- SelectMany the names of length four

```
var names2 = nameList.Where(n => n.Length == 4)
  .SelectMany(n => n);
```

```
M
a
t
t
A
d
a
m
J
o
h
n
O
w
e
n
```

Projection

## SelectMany Example 2

3.51

- We want to create a single sequence of words from a sequence of sentences

```
var sentences = new List<string> {
  "Bob is quite excited.",
  "Jim is very upset."
};
```

- Using SelectMany

```
var words = sentences.SelectMany(
  s => s.TrimEnd('.').Split(' '));
```

```
Bob
is
quite
excited
Jim
is
very
upset
```

- Using LINQ 'from' chaining

```
var words = from s in sentences
            from w in s.TrimEnd('.').Split(' ')
            select w;
```

Projection

## SelectMany Example 3

3.52

- We want to get a flat list of products from categories

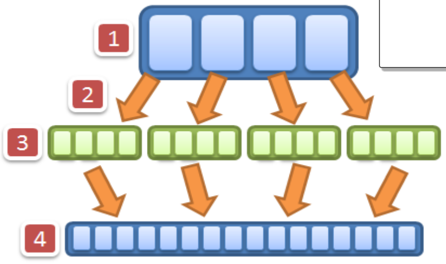
```
NorthwindEntities db = new NorthwindEntities();
```

- Using SelectMany

```
var query = db.Categories.SelectMany(c => c.Products);
```

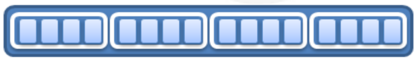
- Using LINQ 'from' chaining

```
var query = from c in db.Categories
            from p in c.Products
            select p;
```



```
... = from c in db.Categories
      select c.Products;
```

Select would give us this:



3.53

Joining and Grouping

## Joining with Query Syntax

- Joining by using where...==

```
var query = from p in db.Products
            from c in db.Categories
            where p.CategoryID == c.CategoryID
            ...
```

- Joining by using join...on...equals

```
var query = from p in db.Products join c in db.Categories
            on p.CategoryID equals c.CategoryID
            ...
```

- Both are equivalent to using the Join extension method (see next slide)

3.54

Joining and Grouping

## Joining with Join extension method

- Join between Categories and Products (1-many)
  - The first lambda chooses property on Category (c) to join on
  - The second lambda chooses property on Product (p) to join on
  - The third lambda expression projects the results, merging properties from each Category entity (cat) and its matching Product entity (prod)

```
var query = db.Categories.Join(db.Products,
  c => c.CategoryID, p => p.CategoryID,
  (cat, prod) => new {
    CategoryID = cat.CategoryID,
    CategoryName = cat.CategoryName,
    ProductName = prod.ProductName });
```

  - One “row” returned for each product (77 in Northwind)

Joining and Grouping

## Joining with GroupJoin extension method

- GroupJoin between Categories and Products (1-many)
  - The first lambda chooses property on Category (c) to join on
  - The second lambda chooses property on Product (p) to join on
  - The third lambda expression projects the results, merging properties from each Category entity (cat) and its matching Product entities (products)

```
var query = db.Categories.GroupJoin(db.Products,
  c => c.CategoryID, p => p.CategoryID,
  (cat, products) => new {
    CategoryID = cat.CategoryID,
    CategoryName = cat.CategoryName,
    NumberOfProducts = products.Count() });
```

  - One “row” returned for each category (8 in Northwind)

Joining and Grouping

## Grouping with Query Syntax

- Groups return List(IGrouping(TKey, TElement))

```
var query = from p in db.Products
            group p by p.Color into colourgroup
            select colourgroup;

List<IGrouping<string, Product>> results = query.ToList();

foreach(IGrouping<string, Product> group in results) {
  listBox1.Items.Add(group.Key); // Red, Blue, etc.
  foreach(Product prod in group) {
    listBox1.Items.Add("   " + prod.ProductName);
  }
}
```

## Tools for Learning
## LINQPad 4 and 101 LINQ Samples

- LINQPad
  - Interactively query databases using LINQ
  - 500 examples

LINQPad
`http://www.linqpad.net`

101 LINQ Samples – C#
`http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b`

Lambda Expressions (C#)
`http://msdn.microsoft.com/en-us/library/bb397687.aspx`

LINQ Query Expressions (C#)
`http://msdn.microsoft.com/en-us/library/bb397676.aspx`

# Module 4
# Creating Classes and Implementing Type-Safe Collections

Creating Classes and Implementing Type-Safe Collections

## Contents

**Exam Topic: Create and implement a class hierarchy**
- ❑ Design and implement an interface (4-11)
- ❑ Create and implement classes based on the IComparable, IEnumerable, IDisposable, and IUnknown interfaces (4-15)

**Exam Topic: Store data in and retrieve data from collections**
- ❑ Choose a collection type (14)
- ❑ Use typed vs. non-typed collections (4-24)
- ❑ Implement custom collections (4-28)
- ❑ Implement collection interfaces (4-26)

## Generics
## What Are They?

4.3

- Define a template for a strongly-typed class
  - Actual type is created at compile time
  - Improves performance and reduces runtime errors
  - Commonly used with collections
- In MSDN documentation Gen(T) means
  - Gen(Of T) for VB
  - Gen<T> for C#

```
class Gen<T, U>
  where T : IComparable
  where U : Person {
  public T Key;
  public U Value;
}
Gen<int, Employee> ga =
  new Gen<int, Employee>();
```

## Generics
## Generic Methods

4.4

- Any type (including non-generic types) can have generic methods
- The generics apply to the types of method signature
  - Specify the types when you call the method

```
class NonGen {
  public void M1<T>
    (T Value) {
    // ...
  }
}

NonGen n = new NonGen();
n.M1<int>(23);
n.M1<string>("Fred");
```

## Generics
## Constraints

- When you define a generic class or method, you can apply restrictions to the kinds of types that can be used

| Constraint | Description |
|---|---|
| where T: struct | The type argument must be a value type |
| where T : class | The type argument must be a reference type |
| where T : new() | The type argument must have a public parameterless constructor; ***must come last*** |
| where T : <base class name> | The type argument must be or derive from the specified base class |
| where T : <interface name> | The type argument must be or implement the specified interface; multiple can be specified; can also be generic |
| where T : U | The type argument supplied for T must be or derive from the argument supplied for U |

Constraints on Type Parameters (C# Programming Guide)
`http://msdn.microsoft.com/en-us/library/d5x73970.aspx`

## Partial Classes
## What Are Partial Classes?

- Allow class definition to be split across multiple files
  - Used by Visual Studio for designer-generated classes

```
partial class Person {
  public string FirstName;
}
```

```
partial class Person {
  public string LastName;
}
```

- VB does not require the keyword on all classes

4.7

## Design Patterns
## You Are Already Using Them

| Pattern | .NET Example |
|---|---|
| Observer, Subject | Events and handlers |
| Iterator | IEnumerable, foreach |
| Decorator | Stream + BufferedStream + CryptoStream |
| Adapter | Using COM with a RCW |
| Factory | WebRequest.Create (HttpWebRequest, FtpWebRequest) |
| Strategy | IComparable, IComparer, LINQ and lambda expressions |
| Composite | Abstract base classes, System.Web.UI.Control |
| Template | Login, GridView, and so on |
| Intercepting Filter | IHttpModule |
| Page Controller | ASP.NET Page class |
| MVC | ASP.NET MVC |
| MVVM | Prism 4 for WPF and Silverlight and Windows Phone |

Discover the Design Patterns You're Already Using in the .NET Framework -
`http://msdn.microsoft.com/en-us/magazine/cc188707.aspx`

4.8

## Design Patterns
## Microsoft patterns & practices

- Recommendations on how to design and develop custom applications using the Microsoft platform
- Categories of patterns & practices offerings
  - Solution Development Fundamentals
  - Cloud Development
  - Desktop Development
  - Phone Development
  - Services Development
  - Web Development

Microsoft® patterns & practices proven practices for predictable results

Solution Development Fundamentals
`http://msdn.microsoft.com/en-us/library/ff921347.aspx`

4.9

Type-Safe Collections

## Queue and Stack

- Queue is a First In First Out (FIFO) data structure
  - Enqueue
  - Dequeue
  - Peek
- Stack is a Last In First Out (LIFO) data structure
  - Push
  - Pop
  - Peek
- They both have generic versions which provide the same capability but strongly-typed
  - Queue<T>
  - Stack<T>

4.10

Type-Safe Collections

## Dictionaries

- Collection of key/value pairs
  - Key must be unique
  - Good for fast lookups based on key
  - Not usually sorted or indexed

| Key | Value |
|---|---|
| | |
| | |
| | |

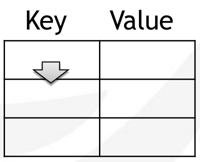
- Common interfaces
  - IDictionary and IDictionary<TKey, TValue>
- Choose based on size of collection
  - Hashtable: >10
  - ListDictionary: <10
  - HybridDictionary: switches between Hashtable & ListDictionary
  - Dictionary<TKey, TValue>: generic dictionary is good for all sizes

Type-Safe Collections

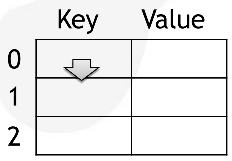
## Specialized Dictionaries

4.11

- Strongly-typed
  - StringDictionary: one string value per string key
  - NameValueCollection: multiple strings per string key, indexed
  - Dictionary<TKey, TValue>: all other types
- Ordered by key
  - OrderedDictionary:
    faster when adding pre-ordered data
  - SortedDictionary<TKey, TValue>:
    faster when adding unordered data



- Indexed and ordered by key
  - SortedList and SortedList<TKey, TValue>
    Smaller but slower than SortedDictionary
  - Do not implement IList but have similar
    methods like RemoveAt(int)



Type-Safe Collections

## LinkedList<T>

4.12

- Each entry points to the entry before and after it
- LinkedList properties and methods
  - First, Last: pointers to these LinkedListNodes
  - AddFirst(newNode), AddLast(newNode), AddBefore(node, newNode),
    AddAfter(node, newNode): all return the new node
  - Remove(node), RemoveFirst(), RemoveLast()
- LinkedListNode instance properties
  - List: pointer to parent LinkedList
  - Previous, Next: pointers to sibling LinkedListNodes
  - Value: value stored in node

Type-Safe Collections

## Other Collection-Related Types

- CaseInsensitiveComparer
- CollectionsUtil: creates case-insensitive collections
  - CreateCaseInsensitiveHashtable()
  - CreateCaseInsensitiveSortedList()
- Abstract classes for custom collections
  - CollectionBase, ReadOnlyCollectionBase, DictionaryBase
- HashSet(T): high-performance set operations
  - IntersectWith(), UnionWith(), IsSubsetOf(), IsSupersetOf(), etc.
- Thread-safe generic collections
  - SynchronizedCollection, SynchronizedKeyedCollection, SynchronizedReadOnlyCollection

Type-Safe Collections

## Summary

| Non-generic collection | Generic equivalent |
|---|---|
| ArrayList | List<T> |
| StringCollection | List<string> |
| Queue, Stack | Queue<T>, Stack<T> |
| SortedList | SortedList<TKey, TValue> |
| Hashtable, NameValueCollection, ListDictionary, HybridDictionary | Dictionary<TKey, TValue> |
| StringDictionary | Dictionary<string, string> |
| OrderedDictionary | SortedDictionary<TKey, TValue> |
| CollectionBase, ReadOnlyCollectionBase | Collection<T>, ReadOnlyCollection<T> |
| n/a | LinkedList<T>, HashSet<T> |

5.1

## Module 5
## Creating a Class Hierarchy by Using Inheritance

5.2

Creating a Class Hierarchy by Using Inheritance
## Contents

**Exam Topic: Create types**
❑ Create value types (structs, enum), reference types, generic types, constructors, static variables, classes

**Exam Topic: Create and implement a class hierarchy**
❑ Inherit from a base class (5-6)

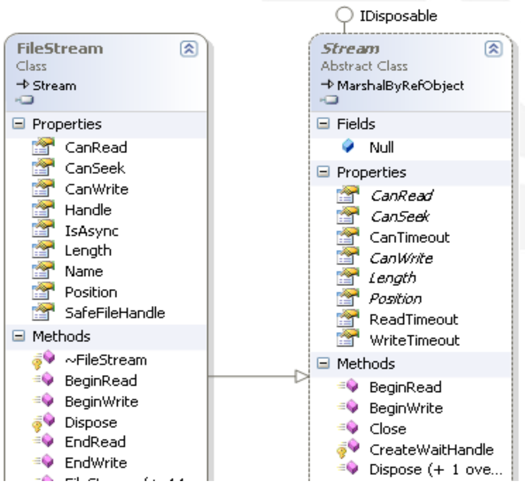
**Exam Topic: Enforce encapsulation**
❑ Enforce encapsulation by using properties, by using accessors (public, private, protected), and by using explicit interface implementation (5-5)

Object-Oriented Programming (C# and Visual Basic)
`http://msdn.microsoft.com/library/dd460654.aspx`

## How to Use a Class Diagram

- Add a Class Diagram to your project
- Drag and drop from Class View

Class Diagram



## Inheritance Keywords

| Keyword | Meaning |
|---|---|
| : | Inherit from one type<br>Implement one or more interfaces |
| override | Change implementation of member; polymorphism supported |
| abstract | Type cannot be instantiated<br>Member must be overridden |
| virtual | Allow member to be overridden |
| new | Replace member even if not overridable; polymorphism not supported |
| base | Instance of base type |
| this | Instance of this type |
| sealed | Type cannot be inherited from |

Object-Oriented Programming

## What Is an Interface?

- A contract; type promises to implement members
- No implementation in the interface itself

```
public interface IMover { // default is internal
  void Move(); // cannot have access modifiers
}
```

- C# types can implement multiple interfaces either implicitly or explicitly:

```
class Car : IMover, IMover2 {
  void Move() { // ...
  void IMover2.Move() { // ...
```

# Module 6
# Reading and Writing Local Data

Reading and Writing Local Data

## Contents

**Exam Topic: Perform I/O operations**
- ❑ Read and write files and streams
- ❑ Read and write from the network by using classes in the System.Net namespace
- ❑ Implement asynchronous I/O operations

**Exam Topic: Serialize and deserialize data**
- ❑ Serialize and deserialize data by using:
  - ❑ binary serialization
  - ❑ custom serialization
  - ❑ XML Serializer
  - ❑ JSON Serializer
  - ❑ Data Contract Serializer

File System and the Registry (C# Programming Guide)
`http://msdn.microsoft.com/library/vstudio/2kzb96fk.aspx`

Streams
## What are Streams?
6.3

- System.IO.Stream represents a stream of bytes
- Backing store streams
  - FileStream, MemoryStream, NetworkStream, …
- Function streams (plug onto other streams)
  - CryptoStream, GZipStream, DeflateStream, BufferedStream, …
- Instance members
  - CanSeek, CanRead, CanWrite, CanTimeout: true/false
  - ReadByte(), WriteByte(byte): work with individual bytes; not as efficient as working with byte arrays
  - Read(byte[], offset, count), Write(byte[], offset, count)
- StreamReader and StreamWriter are helper classes
  - Plug onto streams so you don't have to deal with arrays of byte

Streams
## System.IO.Stream
6.4

- Instance properties
  - Length, Position: long
  - CanSeek, CanRead, CanWrite, CanTimeout: true/false
- Instance methods
  - Seek(long, SeekOrigin): Begin, Current, End
  - Flush(), Close(): ensure anything in buffer is flushed and resources are released
  - ReadByte(), WriteByte(byte): work with individual bytes; not as efficient as working with bytes arrays
  - Read(byte[], offset, count), Write(byte[], offset, count)
- There are helper classes to avoid working with bytes
- When chaining, best to Close the owner of the stream

Streams

## Reading and Writing Files and Streams

- StreamReader is a helper class for streams

```
StreamReader rdr = File.OpenText(@"c:\boot.ini");
while (rdr.Peek() != -1) {
  Console.WriteLine(rdr.ReadLine());
} // or Console.Write(rdr.ReadToEnd());
rdr.Close();
// or Console.WriteLine(File.ReadAllText("@c:\boot.ini"));
```

- StreamWriter is helper class for streams
  - Warning! StreamWriter has an internal buffer, so make sure you Flush or Close before processing the resulting stream

```
StreamWriter writer = File.CreateText(@"c:\somefile.txt");
writer.WriteLine("Hello");
writer.Close(); // flushes the buffer too
```

Streams

## Reading and Writing Text & Binary Files, or Strings

- TextReader (abstract base class)
  - StreamReader: helper class for reading any stream, especially if stream contains text
  - BinaryReader: helper class for reading custom binary data
    - ReadInt32(), ReadBoolean(), and so on
  - StringReader: for reading *strings* in memory; don't confuse with a *StreamReader*
- TextWriter (abstract base class)
  - StreamWriter
  - BinaryWriter
    - Write(...)
  - StringWriter

Streams

## Using a MemoryStream

- StreamWriter is helper class for streams
  - Warning! StreamWriter has an internal buffer, so make sure you Flush or Close before processing the resulting stream
- MemoryStream creates in-memory streams
  - Useful methods: ToArray(), WriteTo(Stream)

```
MemoryStream ms = new MemoryStream();
StreamWriter sw = new StreamWriter(ms);
sw.WriteLine("Hello");
sw.WriteLine("Goodbye");
sw.Flush(); // or Close if you are done
```

Streams

## Using a BufferedStream

- Improve performance by buffering reads and writes
  - FileStream has buffering built-in, but can improve performance of NetworkStream
  - Also use for custom stream implementations

Brad Abrams blog: http://blogs.msdn.com/brada/

- Specify size of buffer in constructor (4096 default)

```
byte[] data = new byte[512];
new Random().NextBytes(data); // random data
```

```
cs = new MyCustomStream(...);
bs = new BufferedStream(cs, 1024))
```

```
bs.Write(data, 0, data.Length);
```

- If you write more than buffer size, it cannot work!

6.9

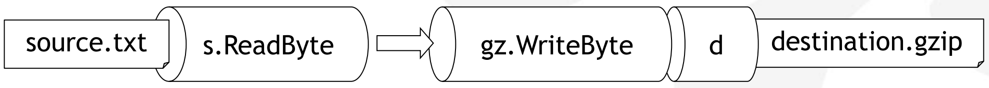
Streams
## Using Compressed Streams

- GZipStream
  - Includes extra header with CRC check
  - Best choice for integrity and compatibility with non-.NET systems
- DeflateStream
  - Uses same algorithm as GZip but no header
  - Best choice for lots of small files which will be compressed and decompressed by .NET applications
- Closing the stream
  - Both own the underlying stream so closing them will close that stream too, unless you pass true as third parameter (next slide)

6.10

Streams
## Using Compressed Streams

```
FileStream s = File.OpenRead("source.txt");
FileStream d = File.Create("destination.gzip");
var gz = new GZipStream(d, CompressionMode.Compress, true);
int i = s.ReadByte(); // returns -1 if EOF
while (i <> -1) {
  gz.WriteByte((byte)i);
  i = s.ReadByte();
}
gz.Close(); // will leave underlying stream (d) open because
            // true was passed into constructor
```



## Streams
## Memory Mapped Files

- Memory mapped files can be used to
  - Efficiently edit very large files
  - Create shared memory for inter-process communication
- System.IO.MemoryMappedFiles
  - Exposes memory mapping functionality provided by Windows API
- System.IO.UnmanagedMemoryAccessor
  - Enables random access to unmanaged memory similar to how UnmanagedMemoryStream enables sequential access to such memory

## Serialization
## What Is It?

- Convert an object into a sequence of bytes for storage or transferral
  - Used behind-the-scenes in other technologies e.g. services
- Common serialization namespaces in .NET
  - System.Runtime.Serialization
    - BinaryFormatter
    - SoapFormatter
    - DataContractSerializer
  - System.Xml.Serialization
  - System.Runtime.Serialization.Json
    - DataContractJsonSerializer
  - System.Web.Script.Serialization

Serialization

## Deserialization

6.13

- Deserialization is the reverse of serialization
  - Deserialize method returns object, so convert it
- With complex objects, the ObjectManager deals with backward and forward references automatically
- Constructors are NOT called during runtime deserialization
  - Runtime deserialization writes directly to memory when deserializing an object
  - Implement IDeserializationCallback.OnDeserialization method to execute initialization code at the end of the process

Serialization

## Classes That Can Be Runtime Serialized

6.14

- Apply the SerializableAttribute to type
  - Enables automatic serialization of ALL fields
  - Apply [NonSerialized] to a field to prevent it being serialized
- Security
  - Private fields are serialized by default so this could be a security hole
  - Code needs SecurityPermission with the SerializationFormatter flag set

```
[Serializable] class CartItem : IDeserializationCallback {
  public int ID, decimal Price, int Quantity;
  [NonSerialized] public decimal Total;
  void OnDeserialization(object sender) {
    Total = Price * Quantity;
  }
}
```

6.15

Serialization
## Version Compatibility

- If you add new members in the future, deserializing a stream created by the previous version will throw an exception because the new member data is missing
  - OptionalFieldAttribute sets the member to null if it is missing

```
[OptionalField] public bool Taxable
```

- Best practice
  - Never remove serialized field
  - Never apply NonSerialized when it wasn't applied earlier
  - Never change names or types of fields
  - For optional fields, set reasonable defaults in OnDeserialization

6.16

Serialization
## Guidelines

- When in doubt, apply [Serializable] to types with fields
  - This attribute is NOT inherited so it must be applied to ALL types in the inheritance hierarchy

```
[Serializable]
public class Animal { private string Name; ... }
```

```
[Serializable]
public class Dog : Animal { ... }
```

- Mark calculated or temporary fields as NonSerialized
- Note: arrays and collections are serializable if the items are serializable

6.17

XML Serialization

## XmlSerializer

- The XmlSerializer can be used to customize the way a type is serialized into XML and so is often used with services to ensure the correct schema of XML
- Mark your type with attributes to customize the process

```
public class Contact
{
  [XmlElement(Name="FName")]
  public string FirstName;
  public string LastName;
  [XmlAttribute(Name="age")]
  public string Age;
}
```

```
<Contact age="23">
  <FName>Fred</FName>
  <LastName>Smith</LastName>
</Contact>
```

6.18

XML Serialization

## How to Control XML Serialization

| | |
|---|---|
| XmlAnyAttribute<br>XmlAnyElement | Members (that return an array) that can contain any unknown XML attributes or elements |
| XmlArray | Members of the array will be generated as members of an XML array |
| XmlArrayItem | Derived types that can be inserted into an array |
| XmlAttribute | Member will be serialized as an XML attribute |
| XmlElement | Member will be serialized as an XML element (default) |
| XmlIgnore | Ignore the member when the class is serialized |
| XmlInclude | The class should be included when generating schemas |
| XmlRoot | Controls the root item name of the class and any namespace |
| XmlText | The property or field should be serialized as XML text |
| XmlType | The name and namespace of the XML type |

XML Serialization

## How to Conform to an XML Schema

- XML Schema Definition tool (xsd.exe)
  - Generates XML schema from classes in a runtime assembly or common language runtime classes from XDR, XML, and XSD files

```
xsd file.xsd {/classes | /dataset}
  [/element:element]
  [/enableLinqDataSet]
  [/language:language]
  [/namespace:namespace]
  [/outputdir:directory]
  [URI:uri]
  [/parameters:file.xml]

xsd {file.dll | file.exe}
  [/outputdir:directory]
  [/type:typename [...]]
  [/parameters:file.xml]
```

XML Serialization

## Other XML-Related .NET Framework SDK Tools

- XML Serializer Generator Tool (sgen.exe)
  - When not used, XmlSerializer generates code and a serialization assembly for each type every time an application is run
  - Pre-generate a serialization assembly to improve performance
  - The following command line creates MyType.XmlSerializers.dll containing a serializer only for type Person

```
sgen /a:MyType.dll /t:Person
```

- Tools for SOAP services
  - wsdl.exe: generate proxy code for XML Web services
  - disco.exe: discover XML Web services

Custom Serialization
## ISerialization

- Implement ISerialization to replace automatic serialization and deserialization process with your own
  - Constructor: called during deserialization
  - GetObjectData method: called during serialization

```
[Serializable] class ShoppingCartItem : ISerializable {
  protected ShoppingCartItem(SerializationInfo info,
      StreamingContext context) {
    productId = info.GetInt32("Product ID");
    // ...
  }
  public virtual void GetObjectData(SerializationInfo info,
      StreamingContext context) {
    info.AddValue("Product ID", productId);
    // ...
  }
}
```

Custom Serialization
## Responding to Serialization Events

- Attributes that can be applied to methods that
  - Accept a StreamingContext parameter and return void
- Four attributes
  - OnSerializing / OnSerialized : before / after serialization
  - OnDeserializing / OnDeserialized : before / after deserialization
- Each attribute can only be applied once, but multiple can be applied to the same method

```
[OnSerializing] [OnSerialized]
void CalculateTotal(StreamingContext sc) {
  total = price * quantity;
}
[OnDeserialized]
void CheckTotal(StreamingContext sc) {
  if (total == 0) CalculateTotal(sc); }
```

6.23

Custom Serialization

## Change Serialization Based on Context

- StreamingContext properties
  - Context: defaults to null, State: defaults to All
  - Must be passed in when constructing a formatter

```
bf = new BinaryFormatter(null, new StreamingContext(
  StreamingContextStates.File |
  StreamingContextStates.Persistence));
```

- State is bit flag indicating the source or destination
  - All: any of the below (default)
  - CrossAppDomain, CrossProcess, CrossMachine: between application domains, processes, or machines
  - File, Persistence, Other : file, database, or unknown destination
  - Remoting: remoting to an unknown destination
  - Clone: copy of the object

6.24

Custom Serialization

## How to Create a Custom Formatter

- Implement IFormatter interface
  - Both BinaryFormatter and SoapFormatter implement it
- FormatterServices provides static help methods

Custom Serialization

## Surrogate Serialization

- Allows a class to serialize another
  - Can therefore serialize a class not marked as serializable
- Must implement ISerializationSurrogate
  - GetObjectData(), SetObjectData()

```
class PersonSurrogate : ISerializationSurrogate { ... }
```

- Must add to surrogate selector for the formatter

```
SurrogateSelector ss = new SurrogateSelector();
ss.AddSelector(typeof(Person),
  new StreamingContext(...), new PersonSurrogate());
BinaryFormatter bf = new BinaryFormatter();
bf.SurrogateSelector = ss;
// serialize as normal
```

Custom Serialization

## Unsafe Deserialization

- UnsafeDeserialize method
  - Only the immediate caller is required to have SerializationFormatter permission
  - In full trust scenarios, UnsafeDeserialize provides better performance than Deserialize
  - Do not use this method if your code can be called from partially trusted code, use Deserialize instead

Serialization

## Summary Comparison

| | | |
|---|---|---|
| **Namespace** | System.Runtime .Serialization | System.Xml .Serialization |
| **Types that perform serialization** | BinaryFormatter SoapFormatter IFormatter | XmlSerializer |
| **Required on your type** | [Serializable] | Parameterless constructor |
| **What gets serialized** | All fields | Public fields and properties |
| **To exclude** | [NonSerialized] | [XmlIgnore] |
| **To customize** | ISerializable, [OnSerializing], etc. | [Xml…], [Soap…] attributes |

Data Contracts

## Serializing Object References

- By default the DataContractSerializer serializes objects by value

```
[DataMember] public SomeClass A = someInstance;
[DataMember] public SomeClass B = someInstance;
```

```
<A>contents of someInstance</A>
<B>contents of someInstance</B>
```

- To instruct the DataContractSerializer to preserve object references, especially for circular references

```
[DataContract(IsReference=true)]
```

```
<A id="1">contents of someInstance</A>
<B ref="1" />
```

Interoperable Object References
http://msdn.microsoft.com/en-us/library/cc656708.aspx

6.29

Data Contracts

## Data Member Default Values

- When a reference type is null, xsi:nil is used in XML

```
[DataMember]
public string FirstName = null;
```

```
<FirstName xsi:nil="true" />
```

- To exclude element when values are equal to defaults

```
[DataMember(EmitDefaultValue=false)]
public int Height = 0;
[DataMember(EmitDefaultValue=false)]
public int Weight = 10;
```

```
<Weight>10</Weight>
```

DataMemberAttribute.EmitDefaultValue - http://msdn.microsoft.com/en-us/library/system.runtime.serialization.datamemberattribute.emitdefaultvalue.aspx

6.30

Data Contracts

## Data Member Order

- Members ordered base type first, then alphabetically

```
[DataMember] public string FirstName;
[DataMember] public string LastName;
[DataMember] public byte Age;
```

```
<Age> ...
<FirstName> ...
<LastName> ...
```

- To order members explicitly

```
[DataMember(Order = 1)] ... FirstName;
[DataMember(Order = 2)] ... LastName;
[DataMember(Order = 3)] ... Age;
```

```
<FirstName> ...
<LastName> ...
<Age> ...
```

- What order would this use?

```
[DataMember] ... FirstName;
[DataMember(Order = 1)] ... LastName;
[DataMember] ... Age;
```

```
<Age> ...
<FirstName> ...
<LastName> ...
```

- Because members without order written first

Data Member Order
http://msdn.microsoft.com/en-us/library/ms729813.aspx

## Data Contracts
## XML Namespaces

6.31

- It is best practice to provide a namespace for your data contracts rather than use the default tempuri.org

```
[DataContract(
  Namespace="http://www.firebrand.com/hr/2012/11")]
public class Employee {
```

- You can do this globally by using the assembly-level attribute ContractNamespace

```
[assembly:ContractNamespace(
  "http://www.firebrand.com/hr/2012/11",
  ClrNamespace = "Firebrand")]
```

Data Contract Names
http://msdn.microsoft.com/en-us/library/ms731045(v=vs.100).aspx

## JavaScriptSerializer
## Deserializing JavaScript Object Notation (JSON)

6.32

- Provides serialization and deserialization functionality for AJAX-enabled applications
  - For when you want to work with JavaScript Object Notation (JSON) in managed code
- To deserialize a JSON string, use the Deserialize or DeserializeObject methods
  - Deserialize(String, Type): Converts a JSON-formatted string to an object of the specified type
  - Deserialize<T>(String): Converts the specified JSON string to an object of type T
  - DeserializeObject: Converts the specified JSON string to an object graph

JavaScriptSerializer Class
http://msdn.microsoft.com/en-us/library/system.web.script.serialization.javascriptserializer.aspx

File System
## Managing Drives

- DriveInfo.GetDrives() static method
  - Returns an array of DriveInfo
- DriveInfo instance properties
  - Name: "C:\"
  - VolumeLabel: ""
  - DriveFormat: "NTFS", "FAT32"
  - DriveType: CDRom, Fixed, Network, NoRootDirectory, Ram, Removable, Unknown
  - AvailableFreeSpace, TotalFreeSpace, TotalSize: long
  - IsReady: true/false
  - RootDirectory: DirectoryInfo instance

File System
## Managing Files and Folders

- DirectoryInfo instance members
  - GetDirectories(): array of DirectoryInfo
  - GetFiles(): array of FileInfo
  - Exists: true/false
  - Create()
- Directory static methods
  - Exists("...")
  - CreateDirectory("..."), Delete("...")
  - GetCurrentDirectory(), SetCurrentDirectory("...")

File System

## Creating, Copying, Moving, and Deleting Files

6.35

### FileInfo instance methods

- Only needs to check permissions once

```
var fi = new FileInfo("...");
StreamWriter sw = fi.CreateText();
FileStream fs = fi.Create();
fi.CopyTo("destination"); fi.MoveTo("destination");
fi.Delete(); fi.Encrypt(); fi.Decrypt();
fs = fi.Open(...); // specify file options (on next slide)
fs = fi.OpenRead(); fs = fi.OpenWrite();
StreamReader sr = fi.OpenText();
```

### File static methods

- Checks permissions on every method call

```
StreamWriter sw = File.CreateText("...");
File.Copy("...", "destination");
StreamReader sr = File.OpenText("...");
```

File System

## File Enumerations

6.36

### FileAccess

- Read, Write, ReadWrite: request these capabilities

### FileMode

- Create: overwrites existing file
- CreateNew: throws exception if file exists
- Open: throws exception if file doesn’t exist
- OpenOrCreate: opens if file exists, else creates
- Append: append to existing file
- Truncate: empty file, then append

### FileShare

- None, Read, Write, ReadWrite, Delete: allow these actions for other processes that access this file

File System

## Enumeration

6.37

- New enumeration APIs for System.IO.Directory and System.IO.DirectoryInfo that return IEnumerable(T)'s instead of arrays which is more efficient because
  - They do not need to allocate a (potentially large) array
  - You can access the first results immediately instead of waiting for the entire enumeration to take place
- New convenience APIs for efficiently reading, writing, and appending lines from/to a text file using IEnumerable(String)
  - Useful in LINQ scenarios where you may want to quickly and efficiently query the contents of a text file and write out the results to a log file without allocating any arrays

File System

## Monitoring the File System

6.38

```
FileSystemWatcher fsw =
  new FileSystemWatcher();
```

```
fsw.Path = "c:\test\";
fsw.IncludeSubdirectories = true;
fsw.Filter = "*.xml";
fsw.NotifyFilter =
  NotifyFilters.FileName | NotifyFilters.LastWrite;
fsw.EnableRaisingEvents = true;
```

```
fsw.Changed += fsw_Changed;

void fsw_Changed(object sender,
  FileSystemEventArgs e)
{
  // e.ChangeType, e.FullPath
}
```

# Module 7
# Accessing a Database

## Accessing a Database
## Contents

| Topic | Slide |
|---|---|
| Overview | 3 |
| EF 4.1 | 9 |
| EF 4.2 & 4.3 | 16 |
| EF 5.0 | 17 |
| LINQ to XML | 18 |
| ADO.NET “Classic” | 23 |

**Exam Topic: Consume data**
- ❑ Retrieve data from a database
- ❑ Update data in a database

**Exam Topic: Query and manipulate data and objects by using LINQ**
- ❑ Read, filter, create, and modify data structures by using LINQ to XML

Editing Data in Your Application
`http://msdn.microsoft.com/library/vstudio/ms171928.aspx`

Connecting to Data in Visual Studio
`http://msdn.microsoft.com/library/vstudio/ms171886.aspx`

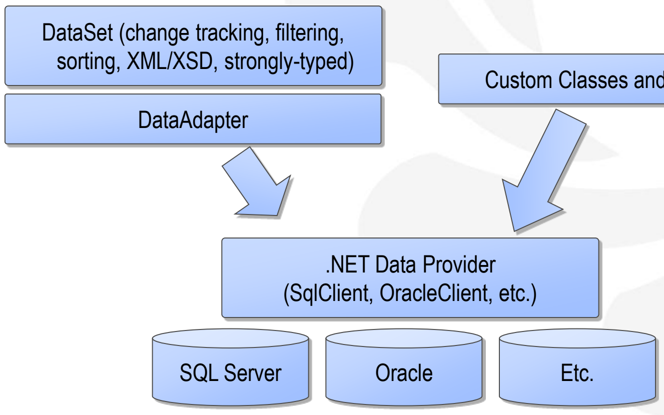
7.3

Overview

## Data APIs in .NET (2002-2007)

### ADO.NET “Classic”

- .NET Framework 1.0, 1.1, and 2.0


DataSet (change tracking, filtering, sorting, XML/XSD, strongly-typed)
DataAdapter
Custom Classes and Collections
.NET Data Provider (SqlClient, OracleClient, etc.)
SQL Server
Oracle
Etc.

* .NET 2.0 adds minor improvements like TableAdapters

7.4

Overview

## Object-Relational Mapping

### What are ORMs?

- Objects are more natural to work with for programmers...
- ...but relational data is better for storage
- Mapping converts CRUD on objects to CRUD on relational data

### Philosophy of ORM

- If you do most work through stored procedures (SELECT, etc.) you will gain very little from using an ORM so use “Classic” ADO.NET instead

### The objects should be “persistence ignorant”

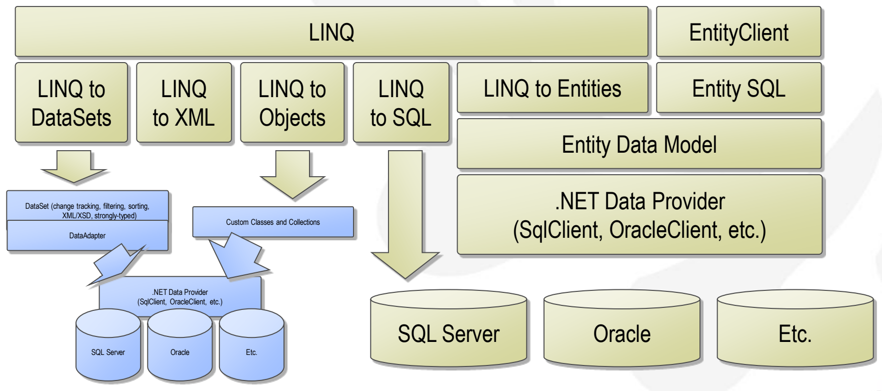
- Members are mostly properties to store column values
- Can have methods for validation and business logic
- Should NOT have methods to store data

Overview

## Data APIs in .NET (2008-2011)

LINQ and Entity Framework

- .NET Framework 3.5 SP1 and .NET 4


LINQ
EntityClient
LINQ to DataSets
LINQ to XML
LINQ to Objects
LINQ to SQL
LINQ to Entities
Entity SQL
Entity Data Model
.NET Data Provider (SqlClient, OracleClient, etc.)
DataSet (change tracking, filtering, sorting, XML/XSD, strongly-typed)
DataAdapter
Custom Classes and Collections
.NET Data Provider (SqlClient, OracleClient, etc.)
SQL Server
Oracle
Etc.
SQL Server
Oracle
Etc.

Overview

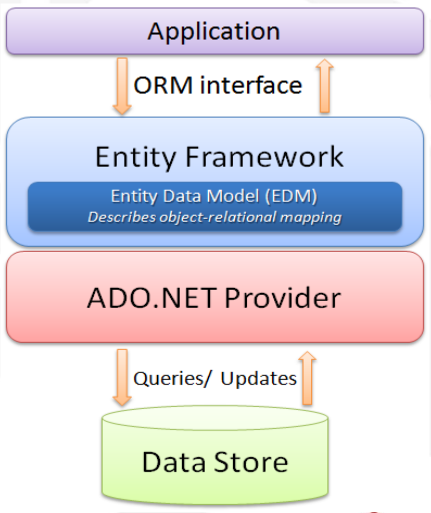
## Entity Framework vs. LINQ to SQL

LINQ to SQL, .NET 3.5

- Created by C# team
- Simple ORM; one-to-one object-to-table mapping (although it does support a discriminator column for simple inheritance scenarios)
- SQL Server only
- Will be supported, but not improved

Entity Framework, .NET 3.5 SP1

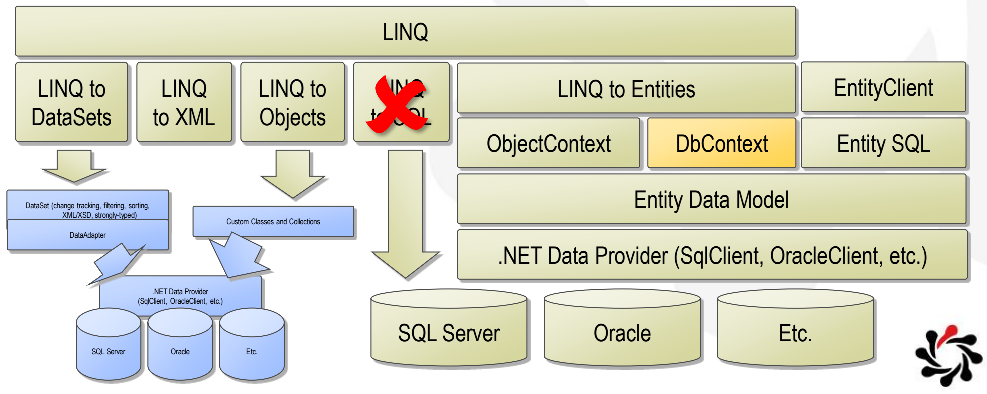
- Created by SQL Server team
- Complex, powerful, flexible ORM
- Heterogeneous data sources
- Future of Microsoft Data APIs



Overview

## Data APIs in .NET (2012+)

- .NET Framework 4.5
  - Appendix A: ADO.NET “Classic” and XML
  - Appendix B: LINQ (and common providers)
  - This Module: Entity Framework


LINQ
LINQ to DataSets
LINQ to XML
LINQ to Objects
LINQ to SQL
LINQ to Entities
EntityClient
ObjectContext
DbContext
Entity SQL
Entity Data Model
.NET Data Provider (SqlClient, OracleClient, etc.)
SQL Server
Oracle
Etc.
DataSet (change tracking, filtering, sorting, XML/XSD, strongly-typed)
DataAdapter
Custom Classes and Collections
.NET Data Provider (SqlClient, OracleClient, etc.)
SQL Server
Oracle
Etc.

Overview

## Data Access APIs: Why Use…

- ADO.NET “Classic”
  - Legacy code, performance, if you mostly use SProcs
- ADO.NET Entity Framework
  - Database or Model First: separate conceptual model from storage model with complex mappings
  - Code First with DbContext: for simple one-to-one mapping models and automatic generation of model or database
- WCF Data Services or ASP.NET Web API OData
  - Expose data via OData (HTTP/REST-architecture service)
- Windows Azure Storage
  - Scalable cloud storage options

EF 4.1

## What is Microsoft ADO.NET Entity Framework 4.1?

7.9

- aka “Magic Unicorn Edition” for VS2010 and later
- EF 4.1 introduces two new features
  - The **DbContext API** is a simplified abstraction over ObjectContext and a number of other types
  - **Code First** is a new development pattern that provides an alternative to the Database First and Model First patterns
- Code First is focused around defining your model using .NET classes
  - These classes can then be mapped to an existing database or be used to generate a database schema
  - Additional configuration can be supplied using Data Annotations or via a fluent API

EF 4.1 Released
`http://blogs.msdn.com/b/adonet/archive/2011/04/11/ef-4-1-released.aspx`

EF 4.1

## Create the Model

7.10

- Create the model using POCO classes

```
public class Category
{
  public string CategoryId { get; set; }
  public string Name { get; set; }
  public virtual ICollection<Product> Products { get; set; }
}

public class Product
{
  public int ProductId { get; set; }
  public string Name { get; set; }
  public string CategoryId { get; set; }
  public virtual Category Category { get; set; }
}
```

EF 4.1

## Create a Context

- Define a context that derives from System.Data.Entity.DbContext and exposes a typed DbSet<TEntity> for each class in my model

```
public class ProductContext : DbContext
{
  public DbSet<Category> Categories { get; set; }
  public DbSet<Product> Products { get; set; }
}
```

- You will need to add a reference to the EntityFramework.dll assembly

EF 4.1

## Mapping to an Existing Database

- The easiest way to point Code First to an existing database is to add a .config connection string with the same name as your derived DbContext

```
<connectionStrings>
  <add name="ProductContext"
    providerName="System.Data.SqlClient"
    connectionString="Data Source=.\SQLEXPRESS;
      Initial Catalog=Products;
      Integrated Security=true;"/>
</connectionStrings>
```

EF 4.1

## Modifying Data

- Use the DbContext

```
using (var db = new ProductContext())
{
  var food = new Category
    { CategoryId = "FOOD", Name = "Foods" };
  db.Categories.Add(food);
  int recordsAffected = db.SaveChanges();
}
```

- If you do not specify a connection string for an existing database then DbContext by convention creates a database for you on localhost\SQLEXPRESS
  - The database is named after the fully qualified name of your derived context

EF 4.1

## Annotations

- You can apply annotations to your model

```
public class Category
{
  [Key]
  public string CategoryId { get; set; }
  [MaxLength(20, ErrorMessage="20 chars max!")]
  public string Name { get; set; }
```

- Annotations include
  - Key, StringLength, MaxLength, ConcurrencyCheck, Required, Timestamp, ComplexType, Column, Table, InverseProperty, ForeignKey, DatabaseGenerated, NotMapped

EF 4.1
## Fluent API

7.15

- Considered a more advanced feature and we would recommend using Data Annotations unless your requirements require you to use the fluent API

```
protected override void OnModelCreating(
  DbModelBuilder modelBuilder)
{
  modelBuilder.Entity<Supplier>()
    .Property(s => s.Name)
    .IsRequired();
}
```

Tutorial: Code First with EF 4.1
`http://codefirst.codeplex.com/`

EF 4.2 and 4.3
## Migration Support

7.16

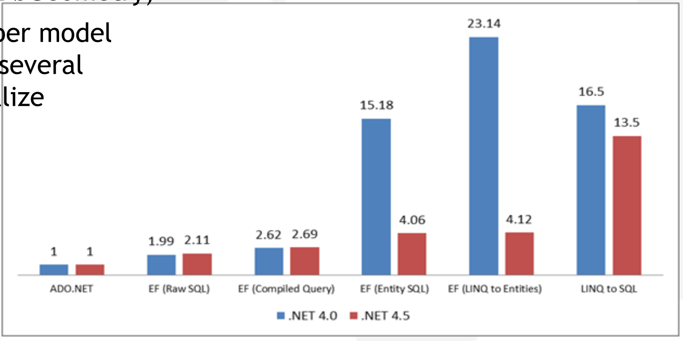
- For example, if you wanted to add a new column to a Blogs table called Url

```
public partial class AddBlogUrl : DbMigration {
  public override void Up() {
    AddColumn("Blogs", "Url", c => c.String());
  }
  public override void Down() {
    DropColumn("Blogs", "Url");
  }
}
```

EF 4.3 Released
`http://blogs.msdn.com/b/adonet/archive/2012/02/09/ef-4-3-released.aspx`

## EF 5.0

- Deployed with .NET Framework 4.5
  - Automatic compilation of LINQ to Entities queries
  - Support for: enums, table-valued functions, spatial data types (DbGeography and DbGeometry)
  - Multiple-diagrams per model allows you to have several diagrams that visualize subsections of your overall model
  - Shapes can have colour applied



Entity Framework 5.0 Performance Improvements
`http://blogs.msdn.com/b/adonet/archive/2012/02/14/sneak-preview-entity-framework-5-0-performance-improvements.aspx`

## LINQ to XML
## Generating an XML File from LINQ-able Entities

- “products” could be an entity set or collection

```
XElement xml = null;

xml = new XElement("Products",
    from p in products
        select new XElement("Product",
            new XElement("ProductID", p.ProductID),
            new XElement("Name", p.Name),
            new XElement("ProductNumber", p.ProductNumber),
            new XElement("Color", p.Color),
            new XElement("Cost", p.Cost),
            new XElement("ListPrice", p.ListPrice),
            new XElement("Size", p.Size)));

xml.Save(productFileName);
```

LINQ to XML

## Generating a Collection from an XML File

7.19

- Convert each child XML element into an entity

```
XDocument doc = XDocument.Load(productFileName);
var query = from product in doc.Descendants("Product")
  select new Product
  {
    ProductID = Convert.ToInt32(
      product.Element("ProductID").Value),
    Name = product.Element("Name").Value,
    ProductNumber = product.Element("ProductNumber").Value,
    Color = product.Element("Color").Value,
    Cost = product.Element("Cost").Value,
    ListPrice = product.Element("ListPrice").Value,
    Size = product.Element("Size").Value
  };
```

LINQ to XML

## Example with Let

7.20

- Imagine that you need to convert this XML into a collection of Car objects

```
<cars>
  <car name="Toyota Coupe">
    <profile name="Vendor" value="Toyota"/>
    <profile name="Model" value="Celica"/>
    <profile name="Doors" value="2"/>
    <support name="Racing" value="yes"/>
    <support name="Towing" value="no"/>
  </car>
  <car name="Honda Accord Aerodec">
    <profile name="Vendor" value="Honda"/>
    <profile name="Model" value="Accord"/>
    <profile name="Doors" value="4"/>
    <support name="Racing" value="no"/>
```

```
public class Car {
  public string Name;
  public string Vendor;
  public string Model;
  public int Doors;
  public bool Racing;
}
```

LINQ to XML

## What Does Let Do?

- let allows you to define local variables in LINQ

```
XDocument xd = XDocument.Load("cars.xml");
var query = from car in xd.Root.Elements("car")
  let profiles =
    from profile in car.Elements("profile")
    select new {
      Name = profile.Attribute("name").Value,
      Value = profile.Attribute("value").Value
    }
  let supports =
    from support in car.Elements("support")
    select new {
      Name = support.Attribute("name").Value,
      Value = support.Attribute("value").Value
    }
...
```

LINQ to XML

## Using let

- ...and then use it in subsequent query clauses

```
...
  select new Car {
    Name = car.Attribute("name").Value,
    Vendor = profiles.Single(
      prof => prof.Name == "Vendor").Value,
    Model = profiles.Single(
      prof => prof.Name == "Model").Value,
    Doors = int.Parse(profiles.Single(
      prof => prof.Name == "Doors").Value),
    Racing = supports.Single(
      sup => sup.Name == "Racing").Value == "yes"
  };
List<Car> cars = query.ToList<Car>();
```

- Another example:

The Linq "let" keyword
http://www.codethinked.com/the-linq-quot3bletquot3b-keyword

ADO.NET “Classic”

## Connections, Commands, DataReaders

- Must open connection before executing commands

```
var con = new SqlConnection(conStr);
var cmd = new SqlCommand(sql, con);
con.Open(); // open connection before executing commands
```

- Common CommandBehaviors
  - CloseConnection, SequentialAccess, SingleResult, SingleRow

```
var reader = cmd.ExecuteReader(CommandBehavior.SingleResult);
while(reader.Read()) // returns true if another row exists
{
  // process row
}
// reader.NextResult(); // returns true if another result exists
```

```
reader.Close(); // close reader before reading parameters
int outputParam = cmd.Parameters[2].Value;
con.Close(); // or use CommandBehavior.CloseConnection
```

# Module 8
Accessing Remote Data

Accessing Remote Data
## Contents

| Topic | Slide |
|---|---|
| OData | 3 |
| WCF Data Services | 7 |
| HTTP Methods | 9 |
| OData .NET Clients | 12 |
| HTTP Clients | 16 |

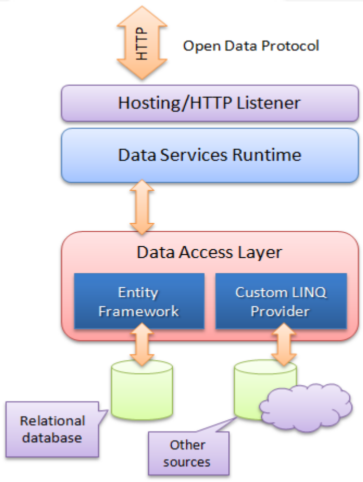
**Exam Topic: Consume data**
❑ Consume JSON and XML data
❑ Retrieve data by using web services

**Exam Topic: Validate application input**
❑ Validate JSON data

OData

## Overview

- OData is a standard for building HTTP services that follow standards for querying the data model
  - It defines a query syntax using URIs similar to SQL
- Two technologies for creating an OData service
  - WCF Data Services (.NET 3.5 +)
  - ASP.NET Web API OData (.NET 4.5 +)



WCF Data Services and OData At-a-Glance
http://msdn.microsoft.com/en-us/data/aa937697

WCF Data Services
http://msdn.microsoft.com/en-us/data/odata.aspx

WCF Data Services Blog
http://blogs.msdn.com/b/astoriateam/

OData

## URL Query Syntax Basics

- To select or order by multiple columns use a comma-separated list

```
http://.../AW.svc/Contacts?
  $select=FirstName,LastName,Age&
  $filter=State eq 'CA' and Price gt 500&
  $orderby=LastName,Age
```

- Case-sensitive!
- Must use $ prefix for keywords
  - $select, $filter, $orderby, $expand
  - $top, $skip
  - /$count: return int
  - $inlinecount: a count is included with the feed
  - $links
  - $metadata

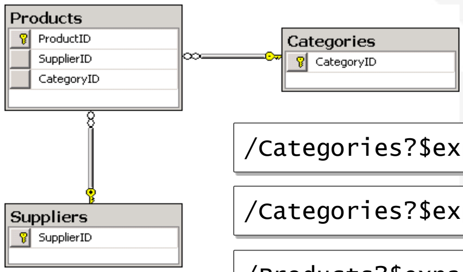
OData: URI Conventions
http://www.odata.org/documentation/uri-conventions#QueryStringOptions

8.5

OData

## $expand

- The syntax of a $expand query option is a comma-separated list of Navigation Properties
  - Each Navigation Property can be followed by a forward slash and another Navigation Property to enable identifying a multi-level relationship



```
/Categories?$expand=Products
```

```
/Categories?$expand=Products/Suppliers
```

```
/Products?$expand=Category,Suppliers
```

Expand System Query Option ($expand)
`http://www.odata.org/documentation/uri-conventions#ExpandSystemQueryOption`

8.6

OData

## URI Query Syntax Examples

| URI | Returns |
|---|---|
| /Customers('ALFKI')/ ContactName | An XML element that contains the ContactName property value for a specific Customer |
| /Customers('ALFKI')/ ContactName/$value | Only the string "Maria Anders" without the XML element |
| /Customers('ALFKI')/Orders | All the orders that are related to a specific Customer |
| /Orders(10643)/Customer | A reference to the Customer entity to which a specific Order entity belongs |
| /Orders?$filter=not endswith(ShipPostalCode,'100') | All the orders the postal codes of which do not end in 100 |
| /Categories(1)/$links/Products | Links to the data instead of the actual data e.g. <uri>http://.../Products(4)</uri> |
| /Categories?$select=Name, Products&$expand=Products | Must select Products if expanding Products |

Accessing Data Service Resources (WCF Data Services)
`http://msdn.microsoft.com/en-us/library/dd728283.aspx`

8.7

WCF Data Services

## How to Create

- In any web project
  - Project – Add New Item – WCF Data Service
  - Create a context class that represents your data
    - ADO.NET Entity Data Model is easiest
    - Or any class that has properties of type IQueryable<T> where T is an “entity” (and optionally implements IUpdatable)
  - Use context class in DataService<TContext>
  - Set permissions

```
public class BlogService : DataService<BlogContext> {
  public static void InitializeService(
                   DataServiceConfiguration config) {
    config.SetEntitySetAccessRule("Blogs", EntitySetRights.All);
    config.SetServiceOperationAccessRule(
      "MyServiceOperation", ServiceOperationRights.All);
    config.DataServiceBehavior.MaxProtocolVersion =
       DataServiceProtocolVersion.V2; } }
```

8.8

WCF Data Services

## Intercepting Queries and Changes

- WCF Data Services enables an application to intercept request messages so that you can add custom logic
  - Define a query interceptor for the Orders entity set

```
[QueryInterceptor("Orders")]
public Expression<Func<Order, bool>> OnQueryOrders()
{
  return o => o.Customer.ContactName ==
        HttpContext.Current.User.Identity.Name;
}
```

  - Check operations to determine type of change

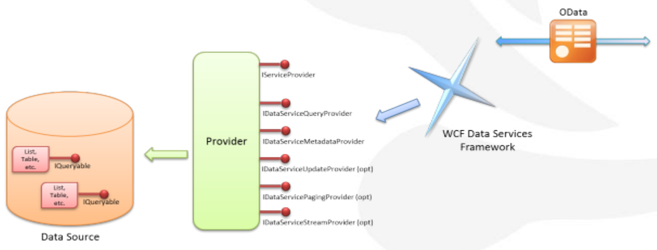
```
[ChangeInterceptor("Products")]
public void OnChangeProducts(
  Product product, UpdateOperations operations)
```

Interceptors (WCF Data Services)
http://msdn.microsoft.com/en-us/library/dd744842.aspx

WCF Data Services

## Custom Data Service Providers

- A Data Service Provider is simply a .NET class that sits between the Data Services Framework and the underlying data source that's being exposed


OData
WCF Data Services Framework
Provider
IServiceProvider
IDataServiceQueryProvider
IDataServiceMetadataProvider
IDataServiceUpdateProvider (opt)
IDataServicePagingProvider (opt)
IDataServiceStreamProvider (opt)
List, Table, etc.
IQueryable
List, Table, etc.
IQueryable
Data Source

```
public class MyDataSource : IUpdatable {
  public IQueryable<Product>Products { get { ...
```

Custom Data Service Providers
http://msdn.microsoft.com/en-us/data/gg191846.aspx

HTTP Methods

## MERGE

- To update a column of a record without overwriting other columns, use MERGE verb and only pass the changed column values

```
MERGE /AW.svc/Contacts(23)
Host: AdventureWorks.com
Content-Type: application-json
{ State: 'CA' }
```

- Use SaveChangesOptions.ReplaceOnUpdate for PUT
- Warning!
  - By default the WCF Data Services client library passes <u>all</u> properties in MERGE, not just the ones that have changed

WCF Data Services: Optimizing bandwidth usage and performance with updates
http://blogs.infosupport.com/wcf-data-services-optimizing-updates-in-the-client-library/

HTTP Methods
## Support for CRUD Operations

- To enable CRUD operations, IIS must allow the following methods on the .svc extension
  - PUT
  - DELETE

HTTP Methods
## X-HTTP-Method

- Some network intermediaries block HTTP verbs like DELETE or PUT or MERGE
  - “Verb tunnelling” or “POST tunnelling” gets around this
- Uses HTTP POST to “wrap” another verb

```
POST /Categories(5)
HTTP/1.1
Host: AdventureWorks.com
X-HTTP-Method: DELETE
```

- To enable on client

```
DataServiceContext.UsePostTunneling = true;
```

2.2.5.8 X-HTTP-Method
http://msdn.microsoft.com/en-us/library/dd541471(PROT.10).aspx

OData .NET Clients

## Loading Related Entities

- DataServiceContext does not support lazy loading so you must use the LoadProperty method to explicitly load related entities

```
context.LoadProperty(order, "LineItems");
foreach(var item in order.LineItems) {
```

- Or use Expand method to pre-load (“eager loading”)

```
... from o in aw.Orders.Expand("LineItems") ...
```

DataServiceContext.LoadProperty Method - http://msdn.microsoft.com/en-us/library/system.data.services.client.dataservicecontext.loadproperty.aspx

OData .NET Clients

## Troubleshooting

- To find out how a LINQ to OData query will translate into an OData URL use RequestUri

```
var query = from p in db.Products
            where p.Color == "Red"
            select p;
string uri =
  ((DataServiceQuery)query).RequestUri.ToString();
```

```
http://localhost:1034/AW.svc/Products()
  ?$filter=Color eq 'Red'
```

OData .NET Clients

## Set Headers in the Client Request

- Create an event handler for SendRequest

```
context.SendingRequest += new EventHandler
  <SendingRequestEventArgs>(OnSendingRequest);
```

- Add the header

```
private static void OnSendingRequest(
  object sender, SendingRequestEventArgs e) {
    // Add an Authorization header that contains an
    // OAuth WRAP access token to the request.
    e.RequestHeaders.Add("Authorization",
      "WRAP access_token=\"123456789\"");
}
```

How to: Set Headers in the Client Request (WCF Data Services)
http://msdn.microsoft.com/en-us/library/gg258441.aspx

HTTP Clients

## async and await work as a pair

- By using the new async and await keywords, you can use resources to create an asynchronous method almost as easily as you create a synchronous method

async modifier, Task<T> return type, Async suffix for name

```
async Task<int> AccessTheWebAsync()
{
  HttpClient client = new HttpClient();
  Task<string> getStringTask =
    client.GetStringAsync("http://msdn.microsoft.com");
  DoIndependentWork(); // executes while async op works
  string urlContents = await getStringTask;
  return urlContents.Length;
}
```

Waits until task is complete, control returns to the caller of AccessTheWebAsync

8.17

HTTP Clients

## WebClient

- Provides common methods for sending data to and receiving data from a resource identified by a URI

| Method | Description |
|---|---|
| DownloadData | Downloads resource as a Byte array from the URI specified |
| DownloadDataAsync | Downloads resource as a Byte array from the URI specified as an asynchronous operation |
| DownloadDataTaskAsync | Downloads resource as a Byte array from the URI specified as an asynchronous operation using a task |
| DownloadFile, … | Downloads resource with the specified URI to a local file |
| DownloadString, … | Downloads the requested resource as a String |
| UploadData, … | Uploads the data (a byte array) as … |
| UploadFile, … | Uploads a local file as … |
| UploadValues, … | Uploads a NameValueCollection as … |

WebClient Class
`http://msdn.microsoft.com/en-us/library/system.net.webclient(v=vs.110).aspx`

8.18

HTTP Clients

## Upload Values

- Uploads the specified name/value collection to the resource identified by the specified URI
  - For an HTTP resource, the POST method is used
  - If the Content-type header is null, the UploadValues method sets it to "application/x-www-form-urlencoded"

```
var myWebClient = new WebClient();
var nvc = new NameValueCollection();
nvc.Add("Name", name);
nvc.Add("Address", address);
nvc.Add("Age", age);
byte[] responseArray = myWebClient.UploadValues(uri, nvc);
// Encoding.ASCII.GetString(responseArray)
```

WebClient.UploadValues Method (String, NameValueCollection)
`http://msdn.microsoft.com/en-us/library/9w7b4fz7.aspx`

9.1

# Module 9
Designing the User Interface for a Graphical Application

9.2

Designing the User Interface for a Graphical Application

## Contents

| Topic | Slide |
|---|---|
| Overview | 3 |
| XAML | 4 |
| What’s Special About WPF? | 10 |
| Layout | 12 |
| Templates | 13 |
| Routed Events | 14 |
| Styles | 18 |
| Diagnostics | 19 |

**Exam Topic: none**

Overview

## What is Windows Presentation Foundation?

9.3

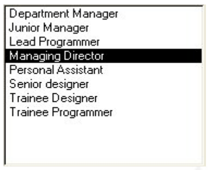
- WPF is a framework to create Windows applications using mark-up (XAML) and code-behind (.cs, .vb)
- Advantages of XAML and WPF
  - Powerful data binding and visualization, media support, 2D and 3D vector graphics, animation, flow and fixed documents
  - Used in Silverlight and Windows 8 Metro (XOML in Workflow)
  - Microsoft is supporting but not extending Windows Forms
- XAML designer in Visual Studio
  - Better XAML IntelliSense, event handling and code writing
- XAML designer in Expression Blend
  - Better visual property, timeline and animation editing

XAML

## Evolution of the List Box

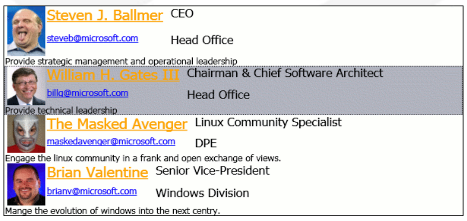
9.4

- 1990s: Visual Basic, C
  - Contains: string (optionally also an integer)
    Displays: string
- Early 2000s: .NET 1.0+
  - Contains: object
    Displays: string (calls *ToString* method)
- 2006 and later: .NET 3.0+
  - Contains: object
    Displays: object



Department Manager
Junior Manager
Lead Programmer
Managing Director
Personal Assistant
Senior designer
Trainee Designer
Trainee Programmer

```
<ListBoxItem>
  <Image ... >
  <Hyperlink ... >
  <RichTextBox ... >
</ListBoxItem>
```



Steven J. Ballmer CEO
steveb@microsoft.com Head Office
Provide strategic management and operational leadership

William H. Gates III Chairman & Chief Software Architect
billg@microsoft.com Head Office
Provide technical leadership

The Masked Avenger Linux Community Specialist
maskedavenger@microsoft.com DPE
Engage the linux community in a frank and open exchange of views.

Brian Valentine Senior Vice-President
brianv@microsoft.com Windows Division
Mange the evolution of windows into the next centry.

## XAML
## What is Extensible Application Markup Language?

- XAML is declarative code
  - Easier for code generators and programmers to read and write
  - Simply instantiates and controls .NET classes
- XAML is an *alternative* to procedural language such as C# and Visual Basic, but is not *required*
  - For example, this XAML could be written using C#

```
<Button Name="b1"
  DockPanel.Dock="Top"> OK
  <Button.Background>
    LightBlue
  </Button.Background>
</Button>
```

```
Button b1 = new Button();
b1.Content = "OK";
b1.Background = new SolidColorBrush(Colors.LightBlue);
DockPanel.SetDock(b1, Dock.Top);
```

## XAML
## Namespaces and Instantiating Objects

- Default defined namespaces

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

- Importing namespaces

```
xmlns:sys="clr-namespace:System:assembly=mscorlib"
xmlns:aw="clr-namespace:Wpf.Examples"
```

- Instantiating objects

```
<sys:Double>98.1</sys:Double>
<aw:Product Name="Bike" ListPrice="12.34" Color="Red" />
```

XAML Namespaces and Namespace Mapping for WPF XAML
http://msdn.microsoft.com/en-us/library/ms747086.aspx

XAML

## Setting Properties in Markup

- Two ways of setting a property in XAML
  - As an attribute or as child element

```
<Button Background="LightBlue">
  ...
</Button>
```

```
<Button>
  <Button.Background>
    LightBlue
  </Button.Background>
  ...
</Button>
```

- Setting default properties

```
<Button Content="Click Me" />
```

```
<Button>Click Me</Button>
```

- Setting attached properties
  - Objects can gain extra abilities by being children of a parent that defines attached properties

```
<Canvas>
  <Button Canvas.Top="20" Panel.ZIndex="1" />
```

XAML

## Setting Attached Properties in Code

- In markup, use the class name that defines the attached property

```
<Canvas>
  <Button Canvas.Top="20" Panel.ZIndex="1" ... />
```

- In code, use static methods on the class that defines the attached property

```
Canvas.SetTop(Button1, 20);
Panel.SetZIndex(Button1, 1);
```

- Can also read attached properties

```
double top = Canvas.GetTop(Button1);
int zindex = Panel.GetZIndex(Button1);
```

9.9

XAML

## Naming Objects

- x:Name or Name (but cannot use both)
  - After x:Name is applied to a framework's backing programming model, the name is equivalent to the variable that holds an object reference or an instance as returned by a constructor
- x:Key
  - Used for items that are being added as values to a dictionary, most often for styles and other resources that are being added to a ResourceDictionary
  - There is actually no corresponding property on the object or even an attached dependency property being set, it is simply used by the XAML processor to know what key to use when calling Dictionary.Add

x:Name Directive
http://msdn.microsoft.com/en-us/library/ms752290.aspx

9.10

What's Special About WPF?

## Dependency Properties and Routed Events

- CLR classes have simple properties and events

```
public string FirstName { get; set; }
public event EventHandler Clicked;
```

- WPF has *dependency* properties and *routed* events
  - Support data binding, styles, resources, animation, event tunnelling and bubbling, and other special features

```
public static readonly DependencyProperty Height =
  DependencyProperty.Register("Height", typeof(int), ...
public static readonly RoutedEvent TapEvent =
  EventManager.RegisterRoutedEvent("Tap",
  RoutingStrategy.Bubble, ...);
```

Dependency Properties Overview
http://msdn.microsoft.com/en-us/library/ms752914.aspx

Routed Events Overview
http://msdn.microsoft.com/en-us/library/ms742806.aspx

## What’s Special About WPF?
## Separation of Control Behaviour and Appearance

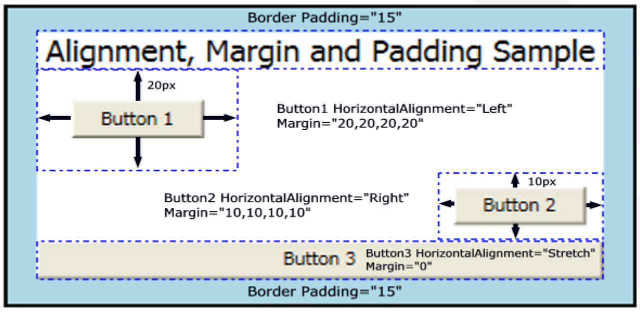
- WPF separates the behaviour of a control from its appearance
  - Every control has a default appearance but this can be replaced
- For example, a button is something that can be clicked to trigger an action
  - Although the default look may be a 3D silver-grey rectangle, a button could look like anything, may be animated, and so on
  - Control templates allow this

## Layout
## Margins and Padding

- Padding is similar to an “inner” Margin in most respects but only used in some classes
  - Block, Border, Control, TextBlock

```
<Border Padding="15"
```

```
myBorder.Padding = new Thickness(15);
```


Border Padding="15"
Alignment, Margin and Padding Sample
20px
Button 1
Button1 HorizontalAlignment="Left"
Margin="20,20,20,20"
10px
Button2 HorizontalAlignment="Right"
Margin="10,10,10,10"
Button 2
Button 3
Button3 HorizontalAlignment="Stretch"
Margin="0"
Border Padding="15"

Alignment, Margins, and Padding Overview
`http://msdn.microsoft.com/en-us/library/ms751709.aspx`

Templates

## Control, Item, and Content Templates

- Template of a Control (instance of a ControlTemplate) decides how a control looks, while the ContentTemplate decides how the Content of the control looks

```
<Button Template={StaticResource A}
        ContentTemplate={StaticResource B} ...
```

```
<ListBox Template={StaticResource C}
         ItemTemplate={StaticResource D} ...
```

```
<Window.Resources>
  <ControlTemplate x:Key="A">
    <ContentPresenter /> ...
```

```
<ControlTemplate x:Key="C">
  <ItemsPresenter /> ...
```

```
<DataTemplate x:Key="B">
  <ContentPresenter Content="{Binding}" />
```

Routed Events

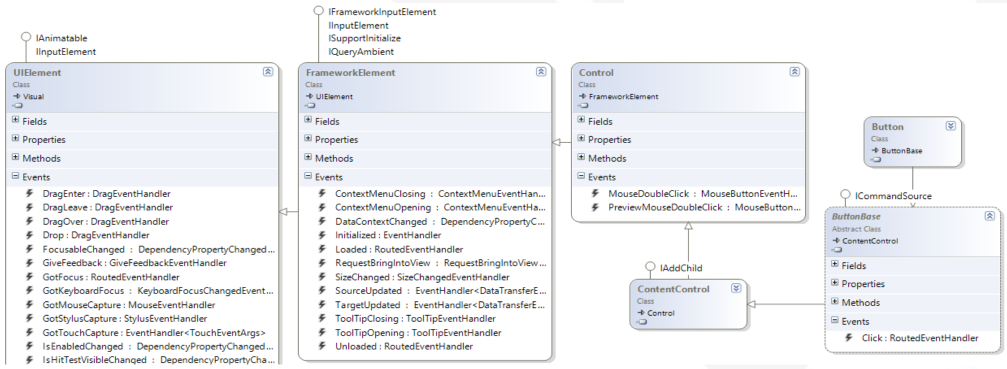
## Tunneling and Bubbling

- How to tell the difference
  - By convention, Preview is a prefix for naming events that are registered to use tunnelling strategy
- Three supported strategies
  - Bubbling
  - Tunnelling
  - Direct

Understand Bubbling and Tunnelling in 5 minutes
http://www.wpfmentor.com/2008/11/understand-bubbling-and-tunnelling-in-5.html

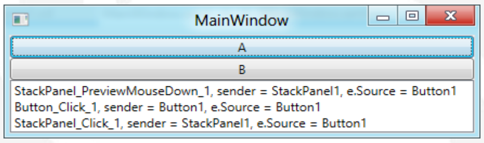
Routed Events
## Inheritance Hierarchy

- UIElement.MouseDown, PreviewMouseDown, and so on
- Control.MouseDoubleClick, PreviewMouseDoubleClick
- ButtonBase.Click


IAnimatable
IInputElement
UIElement
Class
Visual
Fields
Properties
Methods
Events
DragEnter : DragEventHandler
DragLeave : DragEventHandler
DragOver : DragEventHandler
Drop : DragEventHandler
FocusableChanged : DependencyPropertyChanged...
GiveFeedback : GiveFeedbackEventHandler
GotFocus : RoutedEventHandler
GotKeyboardFocus : KeyboardFocusChangedEvent...
GotMouseCapture : MouseEventHandler
GotStylusCapture : StylusEventHandler
GotTouchCapture : EventHandler<TouchEventArgs>
IsEnabledChanged : DependencyPropertyChanged...
IsHitTestVisibleChanged : DependencyPropertyCha...
IFrameworkInputElement
IInputElement
ISupportInitialize
IQueryAmbient
FrameworkElement
Class
UIElement
Fields
Properties
Methods
Events
ContextMenuClosing : ContextMenuEventHan...
ContextMenuOpening : ContextMenuEventHa...
DataContextChanged : DependencyPropertyC...
Initialized : EventHandler
Loaded : RoutedEventHandler
RequestBringIntoView : RequestBringIntoView...
SizeChanged : SizeChangedEventHandler
SourceUpdated : EventHandler<DataTransferE...
TargetUpdated : EventHandler<DataTransferE...
ToolTipClosing : ToolTipEventHandler
ToolTipOpening : ToolTipEventHandler
Unloaded : RoutedEventHandler
Control
Class
FrameworkElement
Fields
Properties
Methods
Events
MouseDoubleClick : MouseButtonEventH...
PreviewMouseDoubleClick : MouseButton...
IAddChild
ContentControl
Class
Control
Button
Class
ButtonBase
ICommandSource
ButtonBase
Abstract Class
ContentControl
Fields
Properties
Methods
Events
Click : RoutedEventHandler

Routed Events
## Difference between *sender* and *e.Source*

- *sender* is the object that handled the event
- *e.Source* is the object that triggered the event



```
StackPanel_PreviewMouseDown_1, sender = StackPanel1, e.Source = Button1
Button_Click_1, sender = Button1, e.Source = Button1
StackPanel_Click_1, sender = StackPanel1, e.Source = Button1
```

```
<StackPanel ButtonBase.Click="StackPanel1_Click"
  PreviewMouseDown="StackPanel1_PreviewMouseDown" ...>
  <Button Content="A" Click="Button1_Click" ...>
  <Button Content="B" ...
```

```
Private Sub StackPanel1_Click( ... )
  ListBox1.Items.Add("StackPanel1_Click, sender = " &
  CType(sender, FrameworkElement).Name & ", e.Source = " &
  CType(e.Source, FrameworkElement).Name)
```

RoutedEventArgs.Source Property – `http://msdn.microsoft.com/en-us/library/system.windows.routedeventargs.source.aspx`

Routed Events
## e.Handled

- Set e.Handled to true to prevent routed events from tunnelling and bubbling to other event handlers
  - Setting e.Handled to true on a button’s Click handler would prevent a parent panel from receiving the Click event…
  - …unless the parent panel added its handler using code and passed true for the handleEventToo parameter

```
AddHandler(Button1.KeyDown, StackPanel1_KeyDown, true);
```

Marking Routed Events as Handled, and Class Handling
http://msdn.microsoft.com/en-us/library/ms747183

Styles
## Defining

- Style using key
  - Control.Property

```
<Style x:Key="myStyle">
  <Setter Property="Control.Background" Value="Blue" />
```

- Style using TargetType
  - Property

```
<Style x:Key="myStyle" TargetType="{x:Type Label}">
  <Setter Property="Background" Value="Blue" />
```

Diagnostics

## Christian Moser’s WPF Inspector

- WPF Inspector is a utility that attaches to a running WPF application to troubleshoot common problems with layouting, databinding or styling
  - Explore a live view of the logical- and visual tree
  - Read and edit property values of elements
  - Watch the data context
  - Debug triggers
  - Trace styles
  - and much more

WPF Inspector
`http://wpfinspector.codeplex.com/`

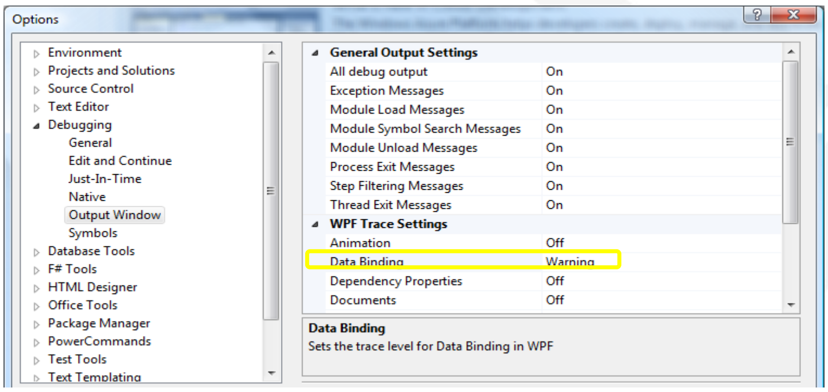
Diagnostics

## Enable Warning Level for Debugging Output

- To see details of PresentationTraceSources

```
xmlns:diag:clr-namespace:System.Diagnostics;assembly=WindowsBase"
```

```
{Binding ..., diag:PresentationTraceSources.TraceLevel=High}
```



Options

- Environment
- Projects and Solutions
- Source Control
- Text Editor
- Debugging
  - General
  - Edit and Continue
  - Just-In-Time
  - Native
  - Output Window
  - Symbols
- Database Tools
- F# Tools
- HTML Designer
- Office Tools
- Package Manager
- PowerCommands
- Test Tools
- Text Templating

| | |
|---|---|
| **General Output Settings** | |
| All debug output | On |
| Exception Messages | On |
| Module Load Messages | On |
| Module Symbol Search Messages | On |
| Module Unload Messages | On |
| Process Exit Messages | On |
| Step Filtering Messages | On |
| Thread Exit Messages | On |
| **WPF Trace Settings** | |
| Animation | Off |
| Data Binding | Warning |
| Dependency Properties | Off |
| Documents | Off |

**Data Binding**
Sets the trace level for Data Binding in WPF

# Module 10
# Improving Application Performance and Responsiveness

Improving Application Performance and Responsiveness

## Contents

| Topic | Slide |
|---|---|
| Performance | 3 |
| Anonymous Delegates | 4 |
| Threading | 5 |
| Tasks | 10 |
| Synchronization | 13 |

**Exam Topic: Implement multithreading and asynchronous processing**
- ❑ Use the Task Parallel library (ParallelFor, Plinq, Tasks)
- ❑ Create continuation tasks
- ❑ Spawn threads by using ThreadPool
- ❑ Unblock the UI
- ❑ Use async and await keywords
- ❑ Manage data by using concurrent collections

**Exam Topic: Manage multithreading**
- ❑ Synchronize resources
- ❑ Implement locking
- ❑ Cancel a long-running task
- ❑ Implement thread-safe methods to handle race conditions

Asynchronous Programming with Async and Await (C# and Visual Basic)
`http://msdn.microsoft.com/library/vstudio/hh191443.aspx`

Overview

## Planning for Application Performance

- Define goals
  - Goals help you to determine whether an application is performing faster or slower
- Understand your platform
  - Always maintain the cycle of measuring, investigating, refining/correcting during your application development cycle
- Make performance tuning an iterative process
  - You should know the relative cost of each feature you will use, for example, reflection is expensive
- Build towards graphical richness
  - Always start with using the least performance intensive resources to achieve your scenario goals; incrementally evolve a UI that adapts to your performance requirements

Delegates

## Anonymous Methods

- In versions of C# before 2.0, the only way to declare a delegate was to use named methods, but C# 2.0 introduced anonymous methods

```
button1.Click += delegate(System.Object o, System.EventArgs a)
  { MessageBox.Show("Click from " + (o as Button).Name); };
```

- With C# 3.0 and later, lambda expressions supersede anonymous methods as the preferred way to write inline code

```
button1.Click += (o, a) =>
  { MessageBox.Show("Click from " + (o as Button).Name); };
```

- Know both for the exam

Anonymous Methods (C# Programming Guide)
http://msdn.microsoft.com/en-us/library/0yw3tz5k(v=vs.110).aspx

Threading

## Manually Managing Threads

- Main thread is foreground thread
  - Keeps process alive
- Background threads do not keep process alive
  - ThreadPool threads are background threads
  - Thread instance has IsBackground property
- A thread can execute a method that conforms to either one of two delegates
  - ThreadStart: no parameter
  - ParameterizedThreadStart: single object parameter
- Call Start method
  - Thread will complete automatically or call Abort

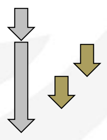
Threading (C# and Visual Basic)
`http://msdn.microsoft.com/library/ms173178.aspx`

Threading

## Asynchronous Processing Model (APM)

- Allows code to run on a different thread so the calling thread isn't blocked
- APM code pattern is used throughout .NET since 1.0
  - Begin... and End... pair of methods e.g. streams have Read method, but also BeginRead and EndRead
  - Example using a callback delegate

```
strm.BeginRead(buffer, 0, buffer.Length,
  new AsyncCallback(CompleteRead), strm);
// do other work
```



```
void CompleteRead(IAsyncResult result) {
  FileStream strm = (FileStream)result.AsyncState;
  int numBytes = strm.EndRead(result);
}
```

Threading

## Delegate Asynchronous Support

10.7

- Even types without built-in support for the Begin+End design pattern can use it through delegates
- Delegate methods
  - BeginInvoke: creates a new thread to execute the method
  - EndInvoke: returns result of method call

```
public int Calc(string s) {
  // method we want to call asynchronously
```

```
delegate int CalcDelegate(string s);
```

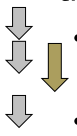
```
CalcDelegate del = new CalcDelegate(Calc);
IAsyncResult iar = del.BeginInvoke("Apples");
// do other work
if(iar.IsCompleted)
  answer = del.EndInvoke(iar);
```

Threading

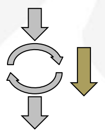
## APM IAsyncResult interface

10.8

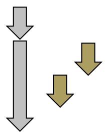
- Members
  - AsyncState: user defined state
  - AsyncWaitHandle: WaitHandle to wait for
  - CompletedSynchronously: how did it complete?
  - IsCompleted: has it finished yet?
- While waiting for the worker thread to complete, the main thread can:



  - Process one chunk of work and then “wait until done”



  - Process multiple, small chunks of work while “polling”



  - Get on with something else and have a callback method called

Threading

## “Wait Until Done” versus “Polling”

- “Wait Until Done” technique

```
FileStream strm = new FileStream("file.txt",
  FileMode.Open, FileAccess.Read, FileShare.Read,
  1024, FileOptions.Asynchronous);
IAsyncResult result = strm.BeginRead(
  buffer, 0, buffer.Length, null, null);
// do some work, then call EndRead to wait until done
int numBytes = strm.EndRead(result);
```

- “Polling” technique

```
IAsyncResult result = strm.BeginRead(
  buffer, 0, buffer.Length, null, null);
while (!result.IsCompleted) {
  // do a small piece of work
}
int numBytes = strm.EndRead(result);
```

Tasks

## Creating and Starting Tasks

- The Task class represents an asynchronous operation

```
var t1 = Task.Factory.StartNew(() => DoAction());
```

```
var t2 = new Task(() => DoAction());
t2.Start();
```

- Other constructors
  - Task(Action, CancellationToken)
  - Task(Action, TaskCreationOptions)
  - Task(Action<Object>, Object): pass in state
  - And other combinations

Task Class
`http://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx`

## Tasks
## Common Members

10.11

| Member | Description |
|---|---|
| AsyncState | Gets the state object supplied when the Task was created, or null if none was supplied |
| IsCanceled, IsCompleted, IsFaulted | Gets whether this Task instance has completed execution due to being canceled, or otherwise |
| ContinueWith(Action<Task>), ContinueWith(Action<Task>, CancellationToken), … | Creates a continuation that executes asynchronously when the target Task completes |
| Delay(Int32), … | Creates a task that will complete after a delay |
| Run(Action), Run(Func<Task>), … | Queues the specified work to run on the ThreadPool and returns a task handle for that work |
| Start(), … | Starts the Task, scheduling it to the TaskScheduler |
| Wait(), … | Waits for the Task to complete execution |
| WaitAll(Task[]), … WaitAny(Task[]), … | Waits for all (or any) of the provided Task objects to complete execution |

## Tasks
## Nested and Child Tasks

10.12

- When code in a task creates a new task and does not specify the AttachedToParent option, the new task is not synchronized with the outer task in any special way
  - Such tasks are called a *detached nested task*
  - The outer task does not wait for the nested task to finish if you call Wait method
- When code in a task creates a new task and DOES specify the AttachedToParent option, the new task is known as a child task of the originating task, which is known as the parent task
  - The outer task DOES wait for the nested task to finish if you call Wait method

Task Parallelism (Task Parallel Library)
`http://msdn.microsoft.com/en-us/library/dd537609.aspx`

## Synchronization
## Thread Access to Shared Resources

10.13

- Multiple threads might access resources simultaneously
- Several types for making your code “thread safe”
  - **Monitor**: exclusive lock for reference types; value types are boxed so will NOT be locked.
    Can also use TryEnter() which uses a timeout to avoid deadlocks

```
lock(this) {
  // ...
}
```

```
Monitor.Enter(this);
try {
  // ...
} finally {
  Monitor.Exit(this);
}
```

  - **ReaderWriterLock**: flag to allow read/write style synchronization; does NOT lock the resource
  - **Interlocked**: exclusive lock for Int32 and Int64
- When synchronizing access to collections, lock the ICollection.SyncRoot property for greater efficiency
  - ...and then check ICollection.IsSynchronized

## Synchronization
## Collections

10.14

- Most .NET collections have an IsSynchronized property that returns false by default
- To create a synchronized collection use the Synchronized static method

```
Hashtable ht1 = new Hashtable();
Hashtable ht2 = Hashtable.Synchronized(ht1);
Console.WriteLine("ht1: {0}", ht1.IsSynchronized);
Console.WriteLine("ht2: {0}", ht2.IsSynchronized);
lock(ht2.SyncRoot)
{
  // enumerate collection
}
```

```
ht1: False
ht2: True
```

- ht1 and ht2 point to same data structure

10.15

Synchronization

## How to Make a Class Thread-Safe

- Most of the code in the base class libraries is NOT thread safe
  - Thread safe means that a type can be safely shared between threads
  - A race condition occurs when a thread pre-empts an operation being performed by another thread causing an error
- The easiest way to make a class thread safe is to lock the whole instance with lock, Monitor.Enter(), or Monitor.TryEnter() methods whenever you are executing code that should not be pre-empted

10.16

Synchronization

## ReaderWriterLock Class

- Does not actually lock anything
  - It is a flag that your code should check before accessing the shared resource
- Can be used to implement common locking pattern
  - Multiple readers can access data at the same time
  - Only one writer at a time (when no readers have locks)
- Readers and writers are queued separately
  - Alternates between a collection of readers, and one writer
- ReaderWriterLockSlim is an improved version

Synchronization
## ReaderWriterLock Useful Members

- Properties
  - IsReaderLockHeld, IsWriterLockHeld
- Methods
  - AcquireReaderLock, AcquireWriterLock
  - UpgradeToWriterLock, DowngradeFromWriterLock
  - ReleaseReaderLock, ReleaseWriterLock
- To avoid deadlocks, the Acquire… methods must supply timeout as milliseconds or TimeSpan
  - -1 milliseconds: infinite, 0: get lock immediately or not at all
- Throws exception after timeout expires

Synchronization
## Interlocked

- Interlocked class works with value types
  - Read(x): safely returns Int64 on 32-bit OS
  - Add(x, y): safely adds y to x (either Int32 or Int64)
  - Increment(x), Decrement(x): works with Int32 and Int64
  - Exchange(x, y): floats, doubles, Int32 and Int64

```
long i;
i++; // unsafe incrementing of 64-bit number
Console.WriteLine(i); // even reading is unsafe!
```

```
Interlocked.Increment(ref i); // safe
Console.WriteLine(Interlocked.Read(ref i)); // safe
```

10.19

Synchronization

## Windows OS Resources

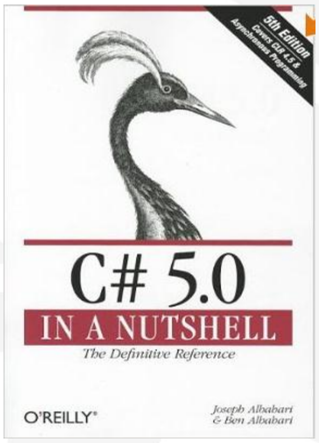
- These are flags ("traffic lights"), not locks
  - Mutex (33x slower than Monitor): synchronization across app domain and process boundaries
  - Semaphore: throttle access to a resource to a set number of threads
  - Event: notify multiple threads that an event has occurred
    - AutoResetEvent and ManualResetEvent classes
  - All classes inherit from WaitHandle class: Handle property, Close, WaitOne methods

```
Mutex m = new Mutex( );
if (m.WaitOne(1000, false)) // wait 1 second for lock
{
    try { // Some work }
    finally { m.ReleaseMutex(); }
}
```

10.20

## Further Study

- Joseph Albahari's free e-book about threading
  - Available online as HTML or downloadable PDF
  - http://www.albahari.com/threading/
- C# 5.0 in a Nutshell: The Definitive Reference
  - By Joseph Albahari and Ben Albahari
  - An excellent additional book because it covers all the topics in the 70-483 exam in depth



# Module 11
# Integrating with Unmanaged Code

Integrating with Unmanaged Code

## Contents

**Exam Topic: Consume types**
- ❑ Handle dynamic types
- ❑ Ensure interoperability with unmanaged code, for example, dynamic keyword

**Exam Topic: Manage the object life cycle**
- ❑ Manage unmanaged resources
- ❑ Implement IDisposable, including interaction with finalization
- ❑ Manage IDisposable by using the using statement
- ❑ Manage finalization and garbage collection

11.3

Memory Management

## What Are Reference Types?

- A reference type is a pointer to an object on the heap
- Assignment copies the memory address on the stack
  - System.String overrides this behaviour to act like a value type but your types should implement the ICloneable interface (provide a Clone method) instead
- Requires garbage collection to remove
  - GC does this automatically when needed
- If your type uses unmanaged resources
  - Finalizer is required: ~*typename*
  - Implement IDisposable is recommended: Dispose()

11.4

Memory Management

## What Is the using Statement?

- What does it do?

```
using (SqlConnection con = new SqlConnection())
{
  // other code
}
```

```
SqlConnection con = new SqlConnection();
try
{
  // other code
}
finally
{
  con.Dispose();
}
```

Memory Management

## GC class

- Controls the system garbage collector, a service that automatically reclaims unused memory

| Method | Description |
|---|---|
| Collect | Forces immediate garbage collection of all generations |
| KeepAlive | References the specified object, which makes it ineligible for garbage collection from the start of the current routine to the point where this method is called |
| ReRegisterForFinalize | Requests that the system call the finalizer for the specified object for which SuppressFinalize has previously been called |
| SuppressFinalize | Requests that the system not call the finalizer for the specified object |
| WaitForFullGCApproach | Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime is imminent |
| WaitForFullGCComplete | Returns the status of a registered notification for determining whether a full, blocking garbage collection by the common language runtime has completed |

Dynamic Types

## Example

- Method chosen at compile-time

```
double x = 1.75;
double y = Math.Abs(x);
```

- Methods chosen at run-time

```
dynamic x = 1.75;
dynamic y = Math.Abs(x); // double
```

```
dynamic x = 2;
dynamic y = Math.Abs(x); // int
```

```
public static class Math
{
  public static decimal Abs(decimal value);
  public static double Abs(double value);
  public static float Abs(float value);
  public static int Abs(int value);
  ...
```

11.7

Dynamic Types

## Comparing static and dynamic typing

- Static

```
Calculator calc = GetCalculator();
double d = calc.Add(2.3, 4.5);
```

- Late-binding using reflection

```
object calc = GetCalculator();
Type calcType = calc.GetType();
double d = (double)calcType.InvokeMember("Add",
  BindingFlags.InvokeMethod, null,
  new object[] { 2.3, 4.5 });
```

- Statically typed to be dynamic

```
dynamic calc = GetCalculator();
double d = calc.Add(2.3, 4.5);
```

11.8

Dynamic Types

## COM Interop

- C# 3.0

```
object fileName = "Test.docx";
object missing  = System.Reflection.Missing.Value;

doc.SaveAs(ref fileName,
    ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing);
```

- C# 4.0 and later

```
doc.SaveAs("Test.docx");
```

## COM Interop
## COM Background

11.9

- COM-compliant components must have an IUnknown interface (AddRef, QueryInterface, Release)
  - Also usually IDispatch for late binding
- Uses type libraries for meta-data (.tlb, .olb)
- Must be registered with OS
  - regsvr32.exe
  - Component (for use at runtime) and type library (for referencing at compile time)
  - Classes and members are identified in registry with GUIDs which should not change between versions

## COM Interop
## Using COM Components

11.10

- Can add a reference in Visual Studio
- Type Library Importer (tlbimp.exe)
  - Generates an assembly (RCW) from a type library; can then be referenced in a project

```
tlbimp MyCOM.tlb
tlbimp MyCOM.tlb /out:MyRCW.dll
csc /r:MyRCW.dll MyApp.cs
```

  - Use /keyfile: or /keycontainer: to apply a strong name to the resulting assembly
- Can also use TypeLibConverter class

COM Interop

## Primary Interop Assemblies

- Unique, vendor-supplied assembly
  - Always use a PIA if available because types have been pre-imported (and optimized)
  - If you import a COM component yourself, you create a set of unique types that are incompatible with those imported by another developer

COM Interop

## Marshal.ReleaseComObject

- Frees the COM object that holds references to resources or when objects must be freed in a specific order
  - Returns the number of remaining references
  - Could construct a loop from which you call this method until the returned reference count reaches zero

P/Invoke Interop

## How to Call Unmanaged DLLs Using DllImport

- P/Invoke calls unmanaged APIs like system DLLs
  - Declare with the DLLImport attribute from System.Runtime.InteropServices
  - Call method as normal .NET method

```
[DllImport("user32.dll")]
static extern IntPtr GetForegroundWindow();
```

P/Invoke Interop

## DllImport

- CharSet
  - Controls how string parameters are marshalled; default is CharSet.Ansi
- EntryPoint
  - Name of function in DLL; only required if you want to use a different name in your code
- ExactSpelling
  - False allows a lookup for multiple possible matches, e.g. GetWindowsPosA or GetWindowsPosW; adds overhead
- SetLastError
  - If true, can get last error, but adds overhead

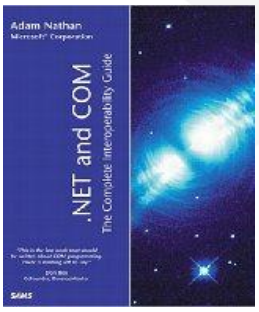
P/Invoke Interop

## Windows Data Types and Structures

- Many P/Invoke calls are to the Windows API
  - So knowing the common Windows data types is useful
- WORD and DWORD
  - 16-bit and 32-bit unsigned integers
- Common Data Types
  - http://msdn.microsoft.com/en-us/library/cc230309(PROT.10).aspx
- Common Data Structures
  - http://msdn.microsoft.com/en-us/library/cc230308(PROT.10).aspx

## Further Study

- .NET 2.0 Interoperability Recipes
  - Bruce Bukovics
  - 632 pages
- .NET and COM: The Complete Interoperability Guide
  - Adam Nathan
  - 1579 pages



# Module 12
# Creating Reusable Types and Assemblies

Creating Reusable Types and Assemblies

## Contents

**Exam Topic: Manage assemblies**
- ❑ Version assemblies
- ❑ Sign assemblies using strong names
- ❑ Implement side-by-side hosting
- ❑ Put an assembly in the global assembly cache
- ❑ Create a WinMD assembly

**Exam Topic: Find, execute, and create types at runtime by using reflection**
- ❑ Create and apply attributes
- ❑ Read attributes
- ❑ Generate code at runtime by using CodeDom and lambda expressions
- ❑ Use types from the System.Reflection namespace (Assembly, PropertyInfo, MethodInfo, Type)

Attributes

## What Are They?

- Meta-data that applies information and functionality to assemblies, types, members

```
[assembly: AssemblyTitle("...")]
[Serializable] [TypeForwardedTo(...)]
Public Class Person
  [FileIOPermission(...)] Public Sub ReadFile()
    ...
```

- Inherit from System.Attribute or derived class
  - Convention is to use …Attribute as suffix, compiler can append suffix automatically when applying attributes

Attributes

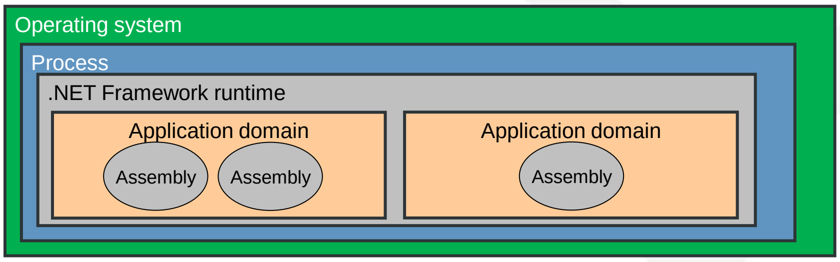
## Common Assembly Attributes

- AssemblyCompany – publishing company name
- AssemblyConfiguration – DEBUG or RELEASE
- AssemblyCopyright – copyright message
- AssemblyCulture – culture for satellite assembly
- AssemblyDescription – a simple description
- AssemblyKeyFile + AssemblyDelaySign – for signing your assembly with a strong name key
- AssemblyVersion – version number of assembly

*GetCustomAttributes reads these values using reflection

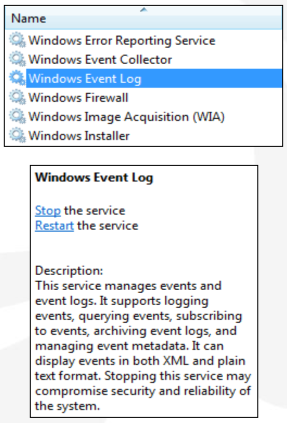
## Application Domains
## What Are They?

- An application domain is a logical container that allows multiple assemblies to run within a single process
  - Prevents direct access to other assemblies' memory
  - More efficient than separate processes
  - Can have different evidence and hence permissions



## Windows Services
## What Are They?

- What is a Windows Service?
  - A long-running process that provides services to other applications, e.g. SQL Server, Exchange Server, Windows Event Log
- Differences to normal .NET processes:
  - Can start as soon as operating system starts
  - Must install before running
  - Cannot debug using F5, must start manually and attach
  - Main method issues Run command
  - User interface interaction is restricted
  - Runs within a security context



| Name |
|---|
| Windows Error Reporting Service |
| Windows Event Collector |
| Windows Event Log |
| Windows Firewall |
| Windows Image Acquisition (WIA) |
| Windows Installer |

**Windows Event Log**

Stop the service
Restart the service

Description:
This service manages events and event logs. It supports logging events, querying events, subscribing to events, archiving event logs, and managing event metadata. It can display events in both XML and plain text format. Stopping this service may compromise security and reliability of the system.

Configuration

## Configuring .NET Applications

- .config files are XML and are processed when starting any .NET application
  - Machine.config
  - My.exe.config or Web.config(s)
- Machine.config is in %Windir%\Microsoft.NET\Framework\v4.0.30319\Config
- Settings can be overridden by subsequent .config files
  - If allowDefinition is MachineOnly they cannot be overridden
  - If allowDefinition is MachineToApplication they can be

Configuration

## Configuration File Example

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

  <appSettings>
    <add key="Foo" value="Hello World!"/>
  </appSettings>

  <connectionStrings>
    <clear/> <!-- clear any defined in Machine.config -->
    <add name="AdventureWorks"
         providerName="System.Data.SqlClient"
         connectionString="..."/>
  </connectionStrings>

</configuration>
```

Configuration

## Using the System.Configuration Namespace

- Reference the System.Configuration assembly
- ConfigurationManager class merges all .config files into read-only collections
  - AppSettings: read from the merged <appSettings>
  - ConnectionStrings: read from the merged <connectionStrings>
  - GetSection: read from any merged section by specifying the path, e.g. "system.web/compilation"
- It can also open specific .config files to enable writing
  - OpenExeConfiguration
  - OpenMachineConfiguration
  - OpenMappedExeConfiguration

Configuration

## External Configuration Sources

- A configuration section can load settings from an external file

```
<pages configSource="pages.config" />
```

- Why?
  - More logical and modular structure
  - File-access security and permissions can be used to restrict access to sections of configuration settings
  - Settings that are not used during application initialization (e.g. connection strings) can be modified and reloaded without requiring an application restart
- If any settings require the application to restart

```
<section name="pages" ...
  restartOnExternalChanges="true" />
```

Configuration
## Reading from Configuration Files
12.11

How to read from <connectionStrings>

- ConnectionString Settings object has a property named ConnectionString that contains the text

```
ConnectionStringSettings css =
  ConfigurationManager.ConnectionStrings["AdventureWorks"];
SqlConnection cn = new SqlConnection(css.ConnectionString);
```

The .config is read when first loading an assembly

- To force a refresh, call RefreshSection method

```
ConfigurationManager.RefreshSection("appSettings");
label1.Text =  ConfigurationManager.AppSettings["colour"];
```

Configuration
## Protecting Configuration Files
12.12

How to protect <connectionStrings> programmatically

```
ConnectionStringsSection s = config.GetSection(
  "connectionStrings") as ConnectionStringsSection;
s.SectionInformation.ProtectSection(
  "RsaProtectedConfigurationProvider");
```

Two providers

- RsaProtectedConfigurationProvider
- DataProtectionConfigurationProvider

To use the same encrypted configuration file on multiple servers, such as a Web farm, only the RsaProtectedConfigurationProvider enables you to export the keys and import them on another server

Reflection

## What Is Reflection?

- If you reference an assembly at compile time, you have direct access to it's types
- If you don't, reflection allows you to
  - Load an assembly at runtime
  - Dynamically read information about all the types
  - Dynamically create an instance of a type and use it's members
  - Dynamically generate new types and assemblies and save them

Reflection

## How to Load Assemblies

- Assembly static methods
  - Load(name): usually load from GAC
  - LoadFile(file): load by filename
  - LoadFrom(path): load by path and filename
  - ReflectionOnlyLoad(name): like Load but read-only
  - ReflectionOnlyLoadFrom(path): like LoadFrom but read-only
  - A .NET 4.5 compiled assembly can call code in an older version assembly but the older assembly will be loaded into the .NET 4.5 assembly's process and executed by the CLR 4.0

```
Assembly a = Assembly.LoadFile("...");
Type t = a.GetType("System.Collections.Hashtable");
ConstructorInfo c = t.GetConstructor(Type.EmptyTypes);
object ht = c.Invoke(new object[] {});
MethodInfo m = t.GetMethod("Add");
m.Invoke(ht, new object[] { 1, "Hello" });
```

Reflection

## MethodBody

- MethodInfo.GetMethodBase returns MethodBody
  - GetILAsByteArray
  - LocalVariables
  - ExceptionHandlingClauses
- ildasm.exe
  - Uses reflection to display IL code for any assembly
- Redgate Reflector
  - Uses reflection to reverse-engineer assemblies to multiple languages

Reflection

## BindingFlags enumeration

| Constant | Description |
|---|---|
| DeclaredOnly | Ignore inherited members |
| FlattenHierarchy | Include declared, inherited, and protected members |
| Instance / Static | Include instance or static members |
| Public | include public members |
| NonPublic | Include protected and internal |

```
foreach (PropertyInfo prop in t.GetProperties())
    Console.WriteLine("{0}", prop.Name);
BindingFlags flags = BindingFlags.Public |
  BindingFlags.NonPublic | BindingFlags.Instance;
foreach (MemberInfo member in t.GetMembers(flags))
  Console.WriteLine("{0}: {1}",
    member.MemberType, member.Name);
```

Reflection

## Getting Type Information

12.17

- Type instance properties
  - Name, Namespace, FullName
  - IsValueType, IsClass
  - IsPublic
  - IsAbstract
  - IsSealed
  - IsFamily (protected)
  - IsAssembly (internal / Friend)
  - IsFamilyOrAssembly (protected internal / Protected Friend)
  - And many more...

Reflection

## Generating Types Dynamically

12.18

- System.Reflection.Emit namespace includes types to create dynamic assemblies
  - AssemblyBuilder
  - ModuleBuilder
  - TypeBuilder
  - ConstructorBuilder
  - MethodBuilder
  - PropertyBuilder
  - FieldBuilder
  - EventBuilder
  - ParameterBuilder
  - ILGenerator
  - EnumBuilder

Assemblies

## Command Line Executable Tools

| EXE | Description |
|---|---|
| gacutil | The Global Assembly Cache tool allows you to view and manipulate the contents of the global assembly cache* and download cache |
| regsvr32 | Registers .dll files as command components in the registry |
| sn | The Strong Name tool helps sign assemblies with strong names; it provides options for key management, signature generation, and signature verification |
| regasm | The Assembly Registration tool reads the metadata within an assembly and adds the necessary entries to the registry, which allows COM clients to create .NET Framework classes transparently |
| csc | You can invoke the C# compiler by typing the name of its executable file (csc.exe) at a command prompt |
| al | The Assembly Linker generates a file that has an assembly manifest from one or more files that are either modules or resource files |

* Windows Installer (MSI) can also be used to install assemblies into the GAC (amongst many other tasks).

WinMD Assemblies

## How to Create

- You can use managed code to create your own Windows Runtime types, packaged in a Windows Runtime component
  - Use your component in Windows Store apps with C++, JavaScript, Visual Basic, or C#
  - Support is designed to be transparent to the .NET Framework programmer, however, when you create a component to use with JavaScript or C++, you need to be aware of differences in the way those languages support the Windows Runtime

Programming in C#: (06) Splitting Assemblies, WinMD, Diagnostics and Instrumentation
http://channel9.msdn.com/Series/Programming-in-C-Jump-Start/Programming-in-C-06-Splitting-Assemblies-WinMD-Diagnostics-and-Instrumentation

Creating Windows Runtime Components in C# and Visual Basic
http://msdn.microsoft.com/en-us/library/windows/apps/br230301.aspx

## Module 13
## Encrypting and Decrypting Data

Encrypting and Decrypting Data

## Contents

**Exam Topic: Perform symmetric and asymmetric encryption**
- ❑ Choose an appropriate encryption algorithm
- ❑ Manage and create certificates
- ❑ Implement key management
- ❑ Implement the System.Security namespace
- ❑ Hashing data
- ❑ Encrypt streams

Encrypting and Decrypting Data
`http://msdn.microsoft.com/library/e970bs09.aspx`

Protecting Data

## Three Techniques

- Encrypt
  - Two-way operation (i.e. can be decrypted)
  - Best choice for data such as credit card numbers
- Hash (integrity check)
  - One-way operation (i.e. cannot create original data from hash)
  - A checksum that is unique to a piece of data to ensure no modification occurs
  - Best choice for data such as passwords
- Sign (authentication check)
  - A digital signature is a value that is appended to electronic data to prove it was created by someone who possesses a specific <u>private</u> key; the <u>public</u> key is used to verify the signature at the receiver's end

Protecting Data

## Three Types of Algorithm

- Non-Keyed
  - Simple to code but weak
- Symmetric Key (aka secret or shared key)
  - Same key on both sides
- Asymmetric Keys
  - Public-private key pair
  - Mathematically linked but cannot derive one from the other

Protecting Data

## Symmetric Encryption

13.5

- Good
  - Fast, large amounts of data
- Bad
  - Need a way to share the key
- OS-Implemented Algorithms (unmanaged code)
  - DES (common but should be avoided)
  - TripleDES
  - RC2 (official replacement for DES)
- Managed Algorithms (supports partially-trusted code)
  - RijndahlManaged, AesManaged
  - Advanced Encryption Standard (AES) is Rijndael with fixed block size and iteration count: best choice

Protecting Data

## SymmetricAlgorithm Base Class

13.6

- All symmetric algorithm implementations derive from System.Security.Cryptography.SymmetricAlgorithm
- Important properties
  - Mode: defaults to CipherMode.CBC (Cipher Block Chaining)
  - LegalKeySizes and LegalBlockSize: array of KeySize; has MaxSize and MinSize and SkipSize
  - KeySize: by default is the largest legal size of key
  - BlockSize: number if bits processed at one time
  - Key: the secret key as a byte array, generated automatically by default, but should be stored or set explicitly
  - IV: initialization vector; like the Key, it is a byte array and must be shared with the decryptor
  - Padding: how to fill remaining bytes in last block

Protecting Data

## SymmetricAlgorithm Base Class

- Important methods
  - CreateEncryptor: creates the object that needs to be passed to a CryptoStream
  - CreateDecryptor: creates the object that needs to be passed to a CryptoStream
  - GenerateIV: generates random IV
  - GenerateKey: generates random key
  - ValidKeySize: returns true for a valid key size

Protecting Data

## How to Establish a Symmetric Key

- Two main ways
  - Use default random key or call GenerateKey method and store resulting key
  - Generate from a password using Rfc2898DeriveBytes or PasswordDeriveBytes classes
    - Also needs a salt value, an IV, and the number of iterations used to generate the key but they have defaults

```
// In practice, the user would provide the password
var password = "P@55w0r]>";
var myAlg = new RijndaelManaged();
byte[] salt = Encoding.ASCII.GetBytes("my salt");
var key As New Rfc2898DeriveBytes(password, salt);
myAlg.Key = key.GetBytes(myAlg.KeySize / 8);
myAlg.IV = key.GetBytes(myAlg.BlockSize / 8);
```

Protecting Data

## Asymmetric Encryption

- Good
  - More secure than symmetric encryption
- Bad
  - Slow, small amounts of data
- Algorithm
  - RSACryptoServiceProvider: encrypt (and also sign!)
    - Name comes from initials of three men who invented it
- How it works
  - Sender uses receiver's public key to encrypt data
  - Receiver uses their private key to decrypt
  - Often combined with symmetric for best of both worlds, for example, HTTPS/SSL

Protecting Data

## How to Encrypt and Decrypt Messages

- Call Encrypt or Decrypt; for both passes
  - Array of bytes containing data to encrypt or decrypt
  - Boolean flag determines if Optimal Asymmetric Encryption Padding should be used (Windows XP and later only)
  - Unlike symmetric, does not use streams, uses byte arrays

```
var messageString = "Hello, World!";
var myRsa = new RSACryptoServiceProvider();
var messageBytes = Encoding.Unicode.GetBytes(messageString);
var encryptedMessage = myRsa.Encrypt(messageBytes, false);
```

```
var decryptedBytes = myRsa.Decrypt(encryptedMessage, false);
Console.WriteLine(Encoding.Unicode.GetString(decryptedBytes));
```

Protecting Data

## Hash and Sign

- Non-Keyed Hash Algorithms
  - Secure Hash Algorithm (SHA) with different hash sizes
    - SHA1 (160 bit), SHA256, SHA384, SHA512
  - MD5: Message Digest 5 (128 bit hash)
- Symmetric Keyed Hash Algorithms
  - HMACSHA1: Hash-based Message Authentication Code (HMAC)
  - MACTripleDES: 8, 16, 24 byte keys; 8 byte hash size (64 bit)
- Asymmetric Keyed Hash and Sign Algorithm
  - Digital Signature Algorithm (DSA)
    - DSACryptoServiceProvider: hash and sign data
    - DSA cannot encrypt!
      Do not confuse with RSACryptoServiceProvider

Protecting Data

## Random Number Generators and Salts

- RNGCryptoServiceProvider class
  - The class can be used to generate a random number for use various types of cryptography and other operations
- Example
  - To store user passwords in the database in a way that they cannot be extracted, the passwords need to be hashed using a one-way hashing algorithm such as SHA1
  - To do so, use the RNGCryptoServiceProvider to create a random salt, append the salt to the password, hash it using SHA1 CryptoServiceProvider class, and store the resulting string in the database along with the salt
  - The benefit provided by using a salted password is making a lookup table assisted dictionary attack against the stored values impractical, provided the salt is large enough

Protecting Data

## How to Compute a Nonkeyed or Keyed Hash

13.13

- A console application that accepts filename argument and computes hash and displays it

```
var hash = new MD5CryptoServiceProvider();
var file = new FileStream(args[0], FileMode.Open, FileAccess.Read);
var reader = new BinaryReader(file);
hash.ComputeHash(reader.ReadBytes((int)file.Length));
Console.WriteLine(Convert.ToBase64String(hash.Hash));
```

- A console application that accepts a password and filename argument and computes hash and displays it

```
var saltBytes = Encoding.ASCII.GetBytes("This is my salt");
var passwordKey = new Rfc2898DeriveBytes(args[0], saltBytes);
var secretKey = passwordKey.GetBytes(16);
var hash = new HMACSHA1(secretKey);
// same as nonkeyed from here
```

Security

## Authenticating and Authorizing Users

13.14

- Authentication
  - Who is the user?
- Authorization
  - What are they allowed to do? Usually based on role membership
- Types in System.Security.Principal
  - IIdentity: authentication of a user
  - IPrincipal: authorization of a user
  - WindowsIdentity: Windows user account
  - WindowsPrincipal: Windows group membership
  - GenericIdentity: application-specific user
  - GenericPrincipal: application-specific group or role membership

Security

## WindowsIdentity Class

- Getting a WindowsIdentity
  - GetCurrent: returns the current user account for the process
  - Impersonate: allows changing of the identity of the process
    - Returns a WindowsImpersonationContext instance; call Undo after performing actions as the new user to revert
- WindowsIdentity properties
  - AuthenticationType
  - IsAnonymous, IsAuthenticated, IsGuest, IsSystem
  - Name, Token
  - User: SID or SecurityIdentifier
  - Groups: array of IdentityReference
    - Use Translate(typeof(NTAccount)).Value to convert to an object with a string for the group names

Security

## WindowsPrincipal Class

- Creating a WindowsPrincipal
  - Constructor takes WindowsIdentity object
  - AppDomain.CurrentDomain.SetPrincipalPolicy links principal to Thread.CurrentPrincipal property
- WindowsPrincipal.IsInRole checks roles
  - WindowsBuiltInRole enum for built in groups
  - String value for custom groups ("domain\VS Developers")
- Can be extracted from current thread
  - But your must first set principal policy (VB does this automatically)

```
AppDomain.CurrentDomain.SetPrincipalPolicy(
  PrincipalPolicy.WindowsPrincipal);
```

Security

## PrincipalPermission class and attribute

- Check the user account declaratively

```
[PrincipalPermission(SecurityAction.Demand,
  Role=@"BUILTIN\Administrators")]
void AdministratorsOnlyMethod()
{
```

- Or imperatively

```
void AdministratorsOnlyMethod()
{
  PrincipalPermission p = new PrincipalPermission(
    null, @"BUILTIN\Administrators", true);
  p.Demand();  // throws SecurityException
```

- Parameters: Authenticated, Name, Role

Security

## Further Study

- Programming .NET Security
  - Adam Freeman & Allen Jones
  - Out of print