Cst 556 Distributed Applications for .NET with Mono

Last Modified February 3, 2019 7:37 am

1. Course Objectives, Prerequisites, and Programming in C# with .NET using Mono
1.a Course Objectives

1.a.1 Motivation

- How can I develop distributed applications with C# for any of the platforms: MacOS, Linux, or Windows?

  - How are serialization, threading, sockets, web service deployment, remoting, clients, Windows services and Web services registration and discovery done in C#?

  - What are Web Services and how do they rely on related standards such as XML, SOAP, and WSDL?

  - How to develop a Console or Web application with a GUI that uses a Web Service?

- Course Outcomes

  - Ability to use the Mono and the C# language to develop distributed applications and services for multiple platforms.

  - Develop ability to analyze a distributed application to determine an appropriate paradigm for realization and deployment.
1.a.2 .NET, Mono, and its Audience

• .NET is the Microsoft platform for client and server Web applications.

• Web Services
  - Services useful to serveral users or applications
  - Multiple cost models: free, subscription, pay-per-use
  - Available across multiple platforms, languages and applications

• Examples
  - Credit card purchases, Mapping and directions, Travel related services such as search, cost, schedules and reservations for lodging, airlines, and rental cars. Retail and services information: Which vendors offer what products, at what price; what’s on sale?

• Mono -- A Cross-Platform, open source .NET framework.
  - Originally developed by Ximian and later acquired by Novell. It is now being led by Xamarian, a subsidiary of Microsoft and the .NET Foundation. One of the aims of Mono is to bring better development tools to linux.
1.a.3 Index to Example Programs

- Section’s C# examples: Section 1.e.13 Example Command Line Program
1.b Course Prerequisites

1.b.1 Developing Distributed Applications

- The material in the course augments material in Ser321 which is a more general introduction to distributed application development. By topic, the following topics are helpful for success in this course:
  - Serialization, as done with Json, or built-in in Java.
  - An Event model and its use in distributed programming.
  - Creating threaded programs and synchronizing shared objects.
  - Internet programming with sockets.
  - Command line program development.
  - Experience with the Ant build tool. This course and its examples do not rely on an IDE, but instead are formulated to run from the command-line using the Ant build tool, for compatibility with other languages (Java & C++ with g++).
1.c Program Development in C# with .NET: Console

1.c.1 Programming with the Console

- Installing Mono provides the C# compiler (msc) and the Mono runtime (mono). The examples will also run with Visual Studio .NET. on Windows (csc). With mono, options can be either -option or /option. With VS, only /.
- C# compiler (msc for the Mono system) syntax
  - msc /t:exe MyClass.cs or mcs -t:exe MyClass.cs
  - msc /? | more for full syntax. Or, man mcs
  - Options you should know include: output, generate xml documentation, addmodule, debug, and library.
  - Compile and execute the sample program: HashtableSvc.cs
- Other compiler example syntax:
  - Create a library for the Person class in Person.cs
    msc /t:library /out:Person.dll Person.cs
  - Create an exe that references the Person.dll and GroupMgr.dll libraries
    msc /reference:Person.dll /reference:GroupMgr.dll GroupMgrClient.cs
  - Note that the libraries must be available to execute GroupMgrClient.exe
1.c.2 Reference for .NET On-Line Documentation

- Mono online documentation: https://www.mono-project.com/docs/
- For emacs, C# mode for syntax specific editing: http://www.cybercom.net/~zbrad/DotNet/Emacs/
1.c.3 Compiler Directives

- **Define and Undefine.** Allows you to define (Undefine) a symbol (does not interfere with program identifiers) that can be used to control compilation.
  - `#define symbol`
  - `#undef symbol`

- **Warning and Error.** Generate a level 1 warning (error) for your application
  - `#warning message`
  - `#error message`

- **if, elseif, else and endif.** Conditionally compile based on expressions.

**Example:**

```cpp
#define Version1
#if Version1
MessageBox.Show("MyProgram Version1");
#elseif Version5
MessageBox.Show("MyProgram Version5");
#else
MessageBox.Show("MyProgram");
#endif
```
1.d  Program Development in C# with .NET: IDE

1.d.1  Programming with Visual Studio .NET

- Rich development environment with a “do-it-all” design philosophy.
- Supports development in several languages. In default installation:
  - Visual Basic
  - C#
  - C++

- Supports development of various application types. In default installation:
  - **Windows Application** - having a Windows user interface
  - **Windows Library** - code to be shared by other applications
  - **Windows Control Library** - UI code library for a Windows Application
  - **ASP .NET Web Application** - Application with a Web User Interface
  - **ASP .NET Web Service** - an XML Web Service for other applications
  - **Web Control Library** - UI code library for a Web Application
  - **Console Application** - Command-line application.
  - **Windows Service** - creating a service to be registered with Windows
1.d.2 The IDE
1.e Getting to Know C#

1.e.1 Classes in C#

- Classes may have class members (static) and/or instance members. The instance object refers to itself as this.

- Like any other user defined type (UDT’s), classes reside in a namespace and may specify a visibility of public or internal to that namespace.
  - Some other UDT’s are structures (struct), enumerations (enum), interfaces and delegates.

- There is a default parameterless constructor that is silently removed if the programmer provides constructors with parameters.

- Members are:
  - Methods and Fields: as in Java.
  - Properties: Accessor and mutator functions (get_XYZ and set_XYZ) allow restrictions on property values.

- Each class member must be defined with a visibility of public, private, protected (within class and subclasses) or internal (to the assembly). The default is internal.
1.e.2 Constructors for a Class

- Defined methods to initialize instances of the Class
  - Methods whose name matches the Class name
  - No return type - it initializes an object
  - May explicitly call parent class’s constructors before doing any initializations of its own.

- Initialization and Inheritance
  - Every constructor implicitly calls parent class’s parameterless constructor before doing any initializations of its own.
  - Exception: the constructor for System.Object does not have a parent class.

- Example - a constructor that calls a parent class’s constructor

```csharp
public class CalculatorWSClient : WSClientGui {
    public CalculatorWSClient():
        base(new string[]{"add","subtract","multiply","divide"}, "Calculator"){
            this.callMethodButt.Click += new System.EventHandler(this.buttonClick);
            this.exitButt.Click += new System.EventHandler(this.buttonClick);
        }
    ...
}
```

- A constructor may call another constructor in the same class as:
  - this(argument1, argument2, ...);
1.e.3 Properties

- C# allows the same approach to fields and properties as Java. A field may be private, if so, the class may choose to provide (set and/or get) access through public methods.

- C# also provides properties with get and/or set functions that may be syntactically abbreviated by the object user. For example,

```csharp
private string name;
public string Name {
    get {
        return name;
    }
    set {
        if ( ! value.Equals("Tim") ) name=value;
    }
}
```

- The property **Name**’s accessor and mutator can be used implicitly with:
  - MyClass mc = **new** MyClass(); //MyClass contains property Name
  - mc.Name = “AnythingButTim”; //calls the set function
  - Console.WriteLine(“MyClass object name is: “+mc.Name); //call get
1.e.4 Attributes in C#

- Attributes are a mechanism for marking classes or class members with special properties or metadata.

- Attributes are special classes in C# and .NET
  - Attributes may be user defined classes that extend `System.Attribute`
  - Predefined .NET Attributes: `ReadOnly`, `SecureMethod`, `NonSerialized`, `Serializable`, `WebMethod`, `WebService`, `ImmutableObject`

- Using Attribute Classes
  - Attributes can mark a `class`, a `method` or a `field` declaration
  - Marking associates properties of an Attribute instance with the class or member

- By convention, Attribute class names end in “Attribute”
  - such as `System.SerializableAttribute`, or `System.Web.Services.WebServiceAttribute`.
  - The Attribute suffix can be omitted in marking a class or class member.
1.e.5 Marking a Class or Member with an Attribute

• Marking a class or member associates a new instance of the Attribute with that class or member.

[WebService]
public class GradeManagerWS {...}

- Uses the parameterless constructor for WebServiceAttribute class to mark GradeManager class as a WebService.

• Marking can use constructors with parameters

[WebMethod (true)]
public double DOverL(double lengthOnWater, double displacement) { //implementation }

- Marks using the WebServiceAttribute constructor with a boolean parmater

public WebMethodAttribute( bool enableSession );

• Marking with positional and named parameters

- Positional parameters must occur first and specify the constructor.

- Named parameters follow and can set values of public read/write properties of the Attribute class.

[RegistryKey(“myKey”, Description=”Value specifies the number of concurrent users”)]
public int maxConcurrentProcesses;
1.e.6 Attributes and Reflection

• Attribute instances (and their properties) are available under program control via `System.Attribute.GetCustomAttributes` method.
  - If you use an Attribute to mark a class member, then the Attribute instance properties are available to your source code through reflection on the member that is marked with the attribute.

• Referencing field attribute properties
  - Suppose the `maxConcurrentProcesses` field in `MarkedClass` has been marked with `RegistryKey` attribute.

```csharp
System.Type type = System.Type.GetType("MarkedClass")
FieldInfo field = type.GetField("maxConcurrentProcesses") { 
foreach (Attribute attr in field.GetCustomAttributes()){ 
    RegistryKeyAttribute keyAttr = attr as RegistryKeyAttribute;
    if(keyAttr != null) { 
        Console.WriteLine("Field name: {0} has keyName: {1} and description: {2}", field.Name, keyAttr.keyName, keyAttr.Description);
    }
}
```

• See the example program `TargetServerAttribute.cs`
1.e.7 Delegates

- C# feature for callbacks. Delegate method is used as a method parameter.
  - In distributed applications, some services may include a “callback” or event service.
  - A client of the service registers interest in knowing when something of interest happens. The client specifies a method to be called when the event occurs.
  - Example events: someone posted a response to my thread, I received a mail message, the pressure in a hydraulic system just exceeded a limit, or a new client connected to a chat server.

- Object interaction via delgates.
  - A client (or handler) object registers interest in knowing about an event.
  - The client object (an event handler) receives a call-back whenever an event occurs in another (event source) object.
  - The event source object maintains a list of all handlers interested in knowing when an event occurs. The event source notifies all handlers when an event occurs, by calling the delegate.
1.e.8 Delegates

- Two aspects of **Delegates**
  - **Registration**
    - Signaling the event and invoking an action by calling the **handler**
  - **Registration** includes
    - Service defines a delegate and methods to handle adding and removing registrations.
    - Client calls the add registration method passing the name of a handler method as the **callback**.

- **Action** causes the **source object (service)** to call all registered **handlers (listeners)**.

![Diagram of Delegates](image-url)
1.e.9 Delegates Example Snippet

- In a **Handler** class the client registers a listener.

```csharp
public void PressHighCallback(int val) { Console.WriteLine(val); }
public static void Main (string[] args) {
    Acl myPressSrc = new Acl();
    Acl.PressCallBack pcb = new Acl.PressCallBack(PressHighCallback);
    myPressSrc.AddPressureListener(pcb); }
```

- **Source Service**

```csharp
class Acl {
    public delegate void PressCallBack(int pressureValue);
    public void AddPressureListener(PressCallBack listener) {...} ...
}
```

- An example
  - Event source object defining the delegate: **PressService.cs**
  - Client registering the call-back: **PressHandle.cs**
1.e.10 Interfaces

• Nearly identical to interfaces in Java
  - define a contract that implementing classes must provide
  - interfaces may derive one or more interfaces
  - classes may derive one class and implement any number of interfaces

• Class specification simply lists a single class to derive and any number of interfaces to implement:

  public class MyClass : System.Exception, MyInter1, MyInter2;

• Here is a simple interface definition. By convention interface names begin with the letter “I”, so they can be distinguished from classes in a class declaration.

  interface ICookieJar {
    void AddCookies(string flavor, int numberOfCookies);
    int GetCookies(string flavor, int numberOfCookies);
    int HowManyLeft();
    int HowManyLeft(string flavor);
  }
1.e.11 Exceptions

- Syntax is nearly identical to Java
  - C# does not require a throws in a method signature of a method that throws but does not handle an exception.
  - User defined exceptions derive System.Exception
- Throwing an Exception object
  - **throw new** Exception (“Oh no, I have no idea what you are requesting!”);
- Example from **HashtableSvc.cs**
  - Try clauses in C# may include a general exception clause which includes no Exception class name.

```csharp
public void addHashEntry(string key, string value) {
    try{
        ht.Add(key, value);
    }catch (System.ArgumentException ex) {
        Console.WriteLine("Will replace old association with new");
        ht.Remove(key); ht.Add(key, value);
        Console.WriteLine(ex.StackTrace);
    }catch {
        Console.WriteLine("somebody caused an exception");
    }finally {
        //code that executes no matter whether the try block succeeds
    }
}
```
1.e.12 C# Supports Automatic Boxing

- In **Java, primitive types are not objects**.
  - Vectors can only store objects (and their extensions). To place an int in a Vector in Java, you must explicitly convert the int to an Integer (Box it).
  - `Vector myVec = new Vector();
    myVec.add(new Integer(10));`
  - The Integer constructor **Boxes** an int, making a heap Integer object.

- In **C#, Boxing is done automatically** for the programmer.
  - The C# compiler recognizes expressions and statements where conversion from primitive to (or from) an object are necessary.
  - `Hashtable ht = new Hashtable();
    ht.Add(“AKey”, 10);`

- Java and C# handle primitive types more efficiently than objects.
  - Primitives in both languages are simple values with no object properties and no garbage collection overhead in their management.
  - C# provides implicit boxing as a convenience for the programmer.
1.e.13 Example Command Line Program

• Here is an example that demonstrates the following:
  - Command line compiling and executing
  - Using C# properties
  - Explicit type casting in C#
  - Using xml documentation comments
  - Handling Exceptions

• The program presents a command line interface for manipulating a hashtable
  - See: HashtableSvc.cs located in the example for this section.

• Compiling from the command line use the following command:
  - msc /doc:Hashtablesvc.xml /out:bin/HashtableSvc.exe HashtableSvc.cs
  - You can also open it up with Visual Studio, where you can generate html documentation.

• See the set of background examples in background.jar
1.f Generating XML Documentation for Your Code

1.f.1 Visual Studio Equivalent of Javadocs

- Source code to XML generation is done by the C# compiler not by a specific tool, such as javadoc.

- Generating XML instead of HTML allows the developer to apply any number of transformations, for example using an xslt stylesheet.

- In the code, use special triple-slash (///) comments rather than double-slash (//) or slash-star (/* ... */) style comments, (all three types are legal comments in the language.)

- After the triple-slash, you are free to use any well-formed XML tags, including a predefined set.

- When compiling the source, specify the xml documentation file, such as:

  - csc /doc:HashtableSvc.xml HashtableSvc.cs

  - You can view the generated XML file with the VS.NET XML viewer

- VS.NET has a Build Comment Web Pages option that will generate a new folder in your project directory holding a number of images and html files built based on the xml documentation comments.
1.f.2 Placing the Comments in Source Code.

- Precede Types with XML documentation comments
  - **Class**, **Delegate** or **Interface**
  - /// <summary>
    /// Class to represent a Cookie Jar with multiple flavors of cookies.
    /// </summary>
    public class CookieJar {

- Precede **members** with XML documentation comments
  - **Field**, **Property** and **Method** definitions

- Precede a **namespace** declaration with XML documentation comments.

- For an example of converting csc compiler generated XML to html see: [background.jar](#)
  - Generate html from XML: `ant genhtml -Dxmlfile="oneannotationxml.xml"`
  - To view the generated html: `ant viewhtml -Dbrowser="path to a browser"`
1.f.3 Predefined XML Documentation Tags

- `<summary>` text summarizing code element `</summary>
- `<c>` or `<code>` text within is to marked as code
- `<example>` used for marking up a code example
- `<exception>` used to document which exceptions a class may throw
- `<list>` used to insert a list into the documentation
- `<param>` describes a given parameter
- `<paramref>` associates a given XML tag with a specific parameter
- `<permission>` documents access permissions for a member
- `<remarks>` build a description for a given member
- `<returns>` document the return value of a member
- `<see>` cross-reference related items
- `<seealso>` build an also-see section
- `<value>` documents a given property