.NET and Advanced C# Topics

April 16, 2008

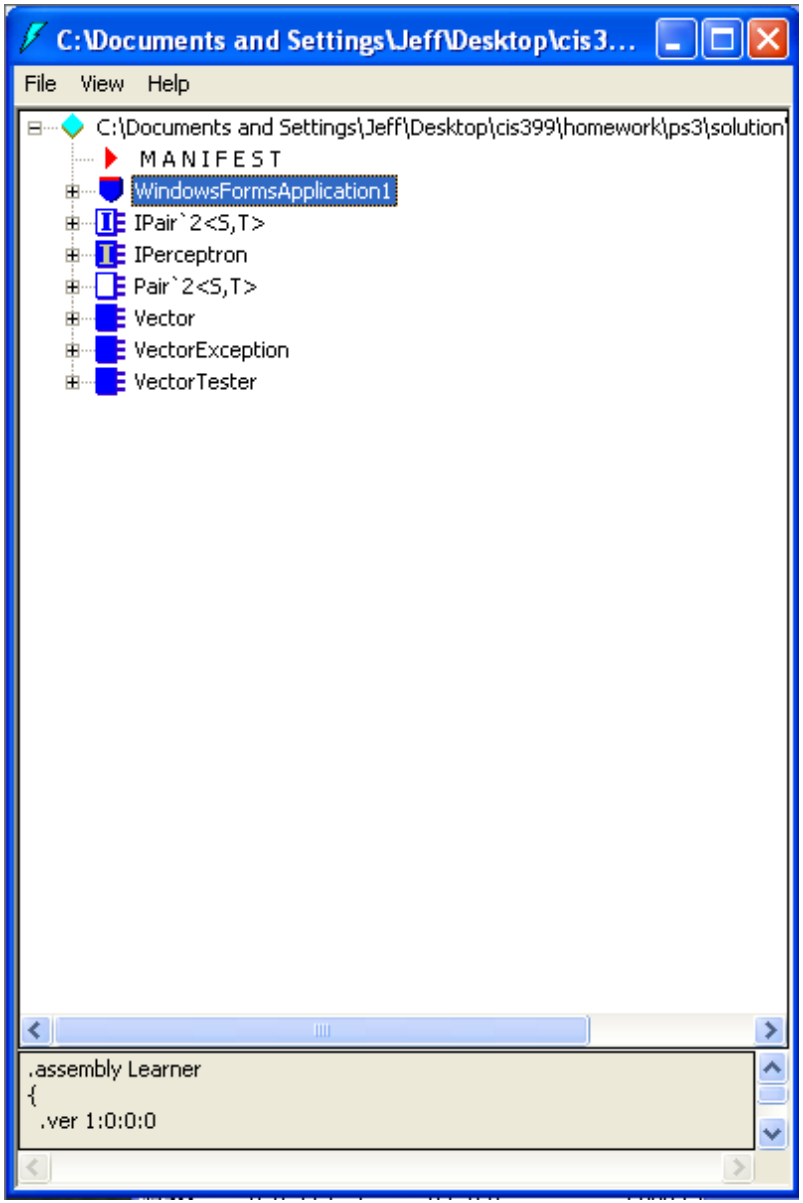
Project Demos

- In class demos
  - Next Wednesday
  - 10 minutes per group
  - Be ready to show off your work and talk a little bit about how C# worked as an implementation language.
  - Avoid disaster: Bring a laptop or test the computer in this room ahead of time. If there's a problem, let me know early!
- Instructor demos
  - Any day next week (send me mail for scheduling)
  - Be prepared to discuss your program's features, and implementation.

Assemblies and modules

- Modules
  - Smallest form of compiler output
  - Contain CIL code and metadata
  - Built using code from a single language
  - Always have extension .netmodule
- Assemblies
  - Correspond to complete libraries (.dll) or programs (.exe)
  - May contain code, metadata, and reference to modules
  - May be written using multiple languages
  - Have distinguished meta data section: the *assembly manifest*

Viewing assemblies in with ildasm



[demo]

Assembly naming

- Two types of names
- Strong names (4 components)
  - **Short name:** e.g. System.XML
  - **Version number:** allows side-by-side execution—a system can contain multiple versions of the same library without conflict
  - **Culture identifier:** allows different code to be loaded based on localization settings
  - **Hash token:** based on on the developer's public key and assembly files names, used to avoid name conflicts between different groups
- Partial names
  - names lacking one of the above components

Code signing

- Can include a digital signature in assembly manifest
- Delayed signing allows code to built without access to private keys
- Prevents malicious parties from swapping assemblies (ensures assembly integrity)
- Does not keep code private (no confidentiality property)

The Global Assembly Cache

- GAC as special directory tree containing assemblies
- Starts at C:\Windows\assembly
- Assemblies in the GAC are accessible to any program
- Assemblies in GAC must be signed or will not load (but this behavior can be disabled)

More on metadata

- Metadata describes types—this information is used to implement reflection
- .Net metadata is extensible—new forms may be added by extending the System.Attribute class

Mixed language programming (Example)

- MyMath.fs (F#, the entire file):

```
let rec fact n = (if n = 0 then 1
                  else n * fact(n-1))
```

- callfact.cs:

```
using System;

public class Caller{
  public static void Main(){
    Console.WriteLine(MyMath.fact(4));
  }

}
```

Common type system

- .Net provides a core set of types that any language can use
  - int, delegate types, etc. . .
- Languages can define nonstandard types to
  - C#: pointer types, sbyte
  - F#: fast functions

Reflection Example

```
using System; using System.Reflection;

public class Reflect {

  public static void Main(){
    Assembly asm =
      Assembly.LoadFrom("RpnCalculator.exe");

    foreach (Type t in asm.GetTypes()){
      Console.WriteLine("Class " + t);

      MethodInfo[] mis = t.GetMethods(
          BindingFlags.Instance | BindingFlags.Public);

      foreach(MethodInfo mi in mis)
        Console.WriteLine("Method " + mi);
    } } }
```

Reflection Example (output)

```
Class RpnCalculator.MainDialog
Method System.String ToString()
Method Boolean ValidateChildren()
Method Boolean ValidateChildren(System.Windows.Form
Method Void RemoveOwnedForm(System.Windows.Forms.Fo
Method Void add_ResizeBegin(System.EventHandler)
Method Void remove_ResizeBegin(System.EventHandler)
Method Void add_ResizeEnd(System.EventHandler)
Method Void remove_ResizeEnd(System.EventHandler)
Method Void SetDesktopBounds(Int32, Int32, Int32, I
Method Void SetDesktopLocation(Int32, Int32)
Method Void Show(System.Windows.Forms.IWin32Window)
Method System.Windows.Forms.DialogResult ShowDialog
Method System.Windows.Forms.DialogResult ShowDialog
Method System.Drawing.Size get_AutoScaleBaseSize()
```

⋮

Reflection capabilities

- Query type and method data
- Query custom attributes
- Call methods (even private ones!)
- Build new types at runtime using System.Reflection.Emit namespace

Reference and Value types

- Two ways to deal with data
- Reference types—declared with class
  - New objects allocated on the *heap* (big)
  - Old objects cleaned by *garbage collection* (slow)
  - Equality defined by reference
- Value types—declared with struct
  - New objects allocated on the *stack* (smaller)
  - Old objects forgotten and overwritten (fast)
  - Equality defined by structure
- *Autoboxing* lets value types be used where reference types are expected.

P/Invoke (Platform invoke)

- The P/Invoke interface is used to call native code libraries.
- The DllImport attribute identifies which library function to call
- Hardest part: “impedance mismatch” between native and .NET types. *MarshalAs* attributes can provide fine-grain control over necessary type conversions.

P/Invoke Example

```
using System;
using System.Runtime.InteropServices;

public static class Beeper{

  [DllImport("User32.dll")]
  static extern Boolean MessageBeep(
                              UInt32 beepType);

  public static void Main(){
    MessageBeep(0);
  }
}
```