Writing Programs Using Mono

Christian Gross
christianhgross@yahoo.ca

O’Reilly Open Source Convention
July 7 - 11, 2003

Introduction: Christian Gross
• Trainer, Consultant, Mentor
  – Open / Shared Source / Binary technologies
  – Technologies (Web Services, Apache / XML, Java, C++, .NET)
• Training
  – http://www.devspace.com
• Author: Upcoming Book
  “Software Engineering Using Java Jakarta”
• Contact: christianhgross@yahoo.ca
Agenda

- Discuss the various platforms
- Outline some Mono specifics
- Demonstrate Mono and some tools

Is .NET Cross Platform Capable

- The real question is if .NET is something that could be cross-platform
  - Of course it could… BUT …

- Will somebody step up to plate and make .NET cross platform
  - There are multiple routes and objectives…
Why Yet Another Platform?

- It is a garbage collected world after all
- Perl, Python, Java, Ruby, etc
  - All languages / environments are garbage collected, resource managed, class library based, etc
- It makes sense to do this because we know what we want
  - We know which subsystems, networking protocols, etc to use
Languages like C and C++ will be reduced in scope and usage
  - Relegated to specialized programming tasks

Moving Your Skills Forward

VC++  C++  Java  J++  VB

MC++  J#  VB.NET  Python

C#  Perl

MONO  MS.NET  Perl/Python
In a Nutshell this means...

- In a Microsoft World (large number of people) C++ is basically dead for application developers
  - System developers have still as of yet unresolved issues
  - C# may replace C++ (C# has many native language constructs)
- C# is the defacto application programming language
  - Others exist, but when in Rome do as the Romans
- Java has its challenges with respect to .NET
  - But Java has one big advantage!
    - It is here, works on multiple platforms
  C#/.NET is still perceived as a Microsoft Only invention
  - Perception is everything?

Cross Platform Theory

- There are three different .NET Open Source implements
  - Rotor: Microsoft Open Source ECMA distribution
    - Not commercial ware
  - Portable.NET: Open Source .NET implementation that is GNU pure
    - Many in the industry have issues with this implementation since it makes commercial software distribution very difficult
    - Implements a traditional compiler approach
  - Mono: Open Source implementation that has the most momentum
    - Built by Ximian
The .NET Framework World

The ECMA C#/CLI Standards
Rotor: The Shared Source CLI

- Rotor came into life in the Fall of 2000 as the Shared Source CLI
  - License is purely educational and non commercial
- Example implementation of full ECMA standards for CLI / C#
- Cross-platform implementation
  - Code builds/runs on FreeBSD and Windows™ XP
  - Code designed to easily port to other platforms
- SSCLI derived from the main product tree and re-integrated
- Shared source CLI implementation differs:
  - JIT and GC replaced with more portable implementations
  - Many Windows-specific features not included: COM interop, WinForms, and other integration
  - Commercial features not included, such as ADO.NET, enterprise services, NGEN (JIT-ahead), and ASP.NET
**MONO**

- **In Words of Miguel de Icazla**
  - Address next generation of Open Source Language support
  - Sharing more code (Write once, use from any language)
  - Advanced development platform
  - Interoperability with Web Services
  - A free unencumbered foundation for development and innovation
  - Wanted to get the benefits of .NET
So What is Mono Really?

- Mono means Monkey in Spanish
- Mono is an implementation of:
  - Common Language Infrastructure
  - C# compiler
  - Class libraries
  - GCC frontend for CIL
- Includes interpreter and JITer
- Mono runs on different UNIX’s and Windows
  - Different platforms
  - Windows is bootstrapped with CYGWIN

Implementation Details

- Mono is C# and .NET similar not .NET identical
  - Some code may port where other may not
- UI code is not programmed using Windows.forms
  - Mono uses GTK# and there is a QT# layer coming as well
- Portability does exist (mostly)
  - ADO.NET is pretty well implemented
  - Windows.Forms exists, but is based on embedding WINE
  - VB.NET is being worked on
  - Web Services is being worked on
  - Enterprise Services is being worked on
  - ASP.NET exists and is being developed in the form of Mono.XSP
What does this mean?

- A Mono developer is != a .NET developer
  - Like a Java J2EE developer != Apache Jakarta developer
- When using Mono do not think of being able to write cross platform in MS.NET
  - Think of being able to write .NET applications that can execute on MS.NET
- All Mono applications can execute on MS.NET
  - Cross platform is an important issue for Mono
  - eg GUI applications are written using GTK# not Windows.Forms

Garbage Collection

- A Class is instantiated into an object, which when not referenced anymore is garbage collected
- There is an easy way and a complicated way to explain the GC (garbage collector)
  - Most chose the difficult way to explain it
  - You allocate, you do not care about freeing it…
- .NET is not GC pure, but oops GC
- .NET GC works and is good
  - GC works using the concept of a directory of references and whether or not a reference is live
  - Reference counting is not used
Garbage Collection

- Garbage Collection is only as smart as the developer
  - If the developer programs sloppy then the program memory will be sloppy and the program will run slowly
  - The best GC is one that is tuned to an individual application
- A programmer should always be aware of what is going on and program intelligently

Garbage Collection (cont.)

- The oops part is when the programmer forgets to do something or does something different
  - Programmers should not be penalized when they decide to do things in other ways
- The idea is to program in chunks of memory which are collected or managed at specific points of time
- In the worst case the GC will collect all loose memory and recycle it
C# Constructors and Destructors

- Constructors are used to initialize resources
- Destructors are used to free resources
  - Destructor terminology is incorrect because it is called Finalization, but the Finalize method is not accessible from C#
  - Destructor tilde syntax is used
  - Resource freeing is not pre-determined nor predictable

```csharp
class LifeCycleSimple
{
    public LifeCycleSimple()
    {
    }
    ~LifeCycleSimple()
    {
    }
}
for( c2 = 0; c2 < 100; c2 ++) {
    LifeCycleSimple cls = new LifeCycleSimple();
}
GC.Collect();
```

Finalization

- Finalization while seeming similar to destructor has more action under the cover
  - Finalization means mark for deletion and the calling of an additional method
    - The extra method call can add quite a bit of overhead
  - A programming convention is to use a “Dispose” or “Close” method
    - Either of these methods clean up the resources

```csharp
class LifeCycle{
    private long _identifier;
    public LifeCycle( long identifier) {
        _identifier = identifier;
    }
    public void Dispose() {
        GC.SuppressFinalize( this);
    }
    ~LifeCycle() { }
}
```
Finalization Design Issues

- Here is a catch
  - Call “Dispose” and suppress finalization
  - But what happens if “Dispose” is missed?
    - Answer is that you create a private method “Dispose” with a true false flag that is called either from the Finalization code or the “Dispose” code
    - Is a bit of “hack”
- My approach is to determine whether or not this is a collected object
  - If collected then forget about dispose and finalization
  - Otherwise write a good base class that manages everything correctly
  - And when necessary implement “IDisposable”
- Make strategic use of the method “GC.Collect”
  - Lessens the hit at the wrong moment
  - Remember when collecting you are hitting the entire process and potentially other request

Weak References

- Weak References makes it possible to define objects that should be kept alive when resources are available, but deleted when resources are scarce
- Weak References could be implemented in the language, but in Rotor (.NET shared source) is implemented using the COM interoperability layer
  - Uses a trick that Weak References are managed using C++, even though C++ is a legitimate way
C# Types

- Two specific types: Value, Reference
- Value data types copies the contents from one location to another
  - Stored on the stack
  - Fixed length
- Reference data types are identical to pointers
  - Are objects where the reference value is stored on the stack, but reference pointer is stored in heap

C# Classes Inheritance

- C# has a single class inheritance and multiple interface inheritance model
  - Interfaces define intentions that have signatures and no scope identifiers
  - Classes can be many things and have many modifiers to

```csharp
interface BaseInterface1 {
    void Method();
}
interface BaseInterface2 {
    void Method();
}
class DerivedClass : BaseClass, BaseInterface1, BaseInterface2 {
    void BaseInterface2.Method() { }
    void BaseInterface1.Method() { }
}
DerivedClass cls = new DerivedClass();
BaseInterface1 intf1 = cls;
BaseInterface2 intf2 = cls;
intf1.Method();
intf2.Method();
// cls.BaseInterface1.Method();
// cls.Method();
```
**C# Structures**

- While classes are straightforward, structures have boxing issues and are misleading because they “act” like classes
- Boxing is the process of converting a value to a reference and then back again
  - Solves the Java problem of explicitly creating an Integer object from the int data type

```csharp
public interface RunningTotal {
    int GetCount();
    void AddItem(int value);
}

public struct Eggbox : RunningTotal {
    private int _eggCount;
    public Eggbox(int initialCount) {
        _eggCount = initialCount;
    }
    public int GetCount() {
        return _eggCount;
    }
    public void AddItem(int value) {
        _eggCount += value;
    }
}

Eggbox eggs = new Eggbox(12);
((RunningTotal)eggs).AddItem(12);
Console.WriteLine("Actual Eggcount is "+eggs.GetCount()); // Is it 12 or 24?
```

---

**C# Constructors**

- Constructors are straightforward, but follow some rules

```csharp
new DerivedClass()

DerivedClass(): this("ss")
DerivedClass(String s): base("ss")

DerivedClass2()
DerivedClass2(String s)

BaseClass()
BaseClass(String s)

ReallyBaseClass()
```

---

Christian Gross christianhgross@yahoo.ca
C# Constructors (cont.)

- Constructors are straightforward, but follow some rules

```csharp
new DerivedClass(“ss”)  
```

```
BaseClass()  
BaseClass( String s)  
```

```
ReallyBaseClass()  
```

```
DerivedClass() : this(“ss”)  
DerivedClass( String s) : base(“ss”)  
```

```
DerivedClass2()  
DerivedClass2( String s)  
```

```
new DerivedClass2(“ ss”)  
```

Inheritance
Calling Sequence
Static Constructors

- Static constructors make it possible to initialize static data members or properties on a runtime basis
  - Catch is that you do not know when the static constructor will be called
  - Static constructor is called just before class is used in any form whatsoever
- Multiple static constructors do not have a calling sequence

```csharp
class StaticClassBase
{   public static int _baseValue;
    static StaticClassBase()
    {   _baseValue = 123;
    }
}
class StaticClass : StaticClassBase
{   public static readonly int _value;
    static StaticClass()
    {   //int some = StaticClassBase._baseValue;
        _value = 1;
    }
}
```

Inheritance 101

- There are many ways to inherit another class or method
  - So let's go through the basics and get more complex with each example
- Simple Inheritance is the process of overriding individual methods that are available in base classes
  - Higher level classes expose new methods, but casting to lower classes gets lower functionality levels

```csharp
class Base
{   public void Method() {   Console.WriteLine( "Base::Method" );
    }
}
class Derived : Base
{   public new void Method() {   Console.WriteLine( "Derived::Method" );
    }
}
```
Inheritance 101 (cont.)

- Virtual function inheritance is the process of hiding and exposing certain functions
  - Virtual are a declaration of derived classes being allowed to modify the lower level functionality
  - But derived classes can decide if they will override or provide new functionality depending on declaration of derived virtual function

```csharp
public virtual void Method1()
public virtual void Method2()
public virtual new void Method1()
public override void Methods2()
```

Inheritance Calling Sequence

```
Derived cls = new Derived();
Base baseCls;
cls.Method1();
cls.Method2();
baseCls = cls;
baseCls.Method2();
baseCls.Method1();
```

Inheritance 101 (cont.)

- Abstract classes and functions are used to indicate that they must be implemented in a derived class
  - Abstract classes cannot be instantiated directly
  - Abstract methods But derived classes can decide if they will override or provide new functionality depending on declaration of derived virtual function

- Abstract follows the inheritance rules defined thus far

```csharp
abstract class AbstractBase{
    public void Method() {}
    public abstract void MustImplement();
}
class AbstractDerived : AbstractBase{
    public new void Method() {}
    public override void MustImplement() {}
}
AbstractDerived absDerived = new AbstractDerived();
AbstractBase absBase;// = new AbstractBase();
absBase = absDerived;
absBase.Method();
absDerived.Method();
```

Method calls obey the same rules (exception “MustImplement”)

Abstract class cannot be directly instantiated
Inheritance 101 (cont.)

• Sealed is used to ensure that no other class overrides or derives from a class or function
  – Sealed on an individual function only works if something has been overridden in the first place
  • To declare a method and stop it from being overridden use a non-virtual

```csharp
sealed class SealedBase {
    public void Method() {
    }
}
class SealedBase2 {
    public virtual void Method() {
    }
}
class SealedClass : SealedBase2 {
    public sealed override void Method() {
    }
}
```

Class cannot be derived from
Method must be declared as virtual
Method is sealed and cannot be overriden

Inheritance Notes

• As shown in the first example interfaces are like abstract classes except that they have absolutely no implementation
  – Cannot contain fields or other data members
• It may seem that there are way too many ways of inheriting and extending functionality
• The reason for this is because of component versions
  – Two different versions of functions with different names will conflict because the various components cannot compile
  – Programming languages like Java and C++ do not account for components and versions of components and the interactions they may have with each other
In Conclusion

• Mono is a good platform for developing C# and .NET type applications that execute on multiple platforms
  – MS.NET is from Microsoft and runs on the Windows platform
• The trick is to start with Mono and then move to .NET framework
• .NET from Microsoft != Mono
  – MS has thousands of developers
    • People sold to MS are happy with Windows and everything else
  – Mono will do its own thing!
    • Smartest long term survival for Mono is to develop its own identity

Thanks

Questions?