An Experienced Programmer's Guide to C# and the .NET Platform

Michael Stiefel
Reliable Software, Inc. (www.reliablesoftware.com)
development@reliablesoftware.com
co-author "Application Development Using C# and .NET"

Assumptions:

- You know how to code in some “high level” language.
- You want to understand how to develop in .NET, not just see language features.

Caveat:

Understand the object-oriented programming paradigm.

Design Patterns
Programming to an Interface not an Implementation
When to Use Inheritance
When to Use Composition

Serialization Example

Illustrate use of C# with a simple, common, programming task of saving and restoring data.

1. Two customer objects are created
2. Objects are added to a collection.
3. Collection is saved to disk.
4. Collection is restored from disk.

See *Serialize.cs*

No code was written to save or restore the data, only the serialized format, and the destination was specified.

Attributes

The class to be saved is marked with the Serializable ***attribute***.

```
[Serializable]
class Customer
{
    public string name;
    public long id;
}
```

This attribute, along with the object’s layout is added to the ***metadata*** associated with the object.

Metadata

- .NET compilers emit code and metadata
- Metadata contains type information
  - Name, visibility
  - Fields, Methods, Properties, Events
  - Layout (not byte location)
  - Attributes (like Serializable)
- Metadata can be queried
- Stored with code (self –describing data)

Attribute/Aspect Based Programming

- Customer class has Serializable attribute.
  - Serialize method uses metadata to save collection and its members.
- Support intertwined in an application that can’t be placed in a component (behavioral metadata).
  - Support for transactions
  - Security settings
  - Multithreading synchronization

Framework Class Library

- **Console**, **ArrayList**, **FileStream**, **SoapFormatter** are FCL Classes.
- Examples:
  - Networking
  - Security
  - Diagnostics
  - I/O
  - Database
  - XML
  - Web services and Web programming
  - Windows User Interface

Namespaces

FCL classes are divided into namespaces to help resolve name conflicts.

```
using System;
using System.Collections;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Soap;
```

You can define your own namespaces.

Garbage Collection

- Memory was never deallocated.
- Memory that passes out of scope or is orphaned is placed on a list of memory locations that can be periodically reclaimed.
  - Produces fast memory allocation and deallocation
- Eliminates memory leaks.
- **Cust**, **list** are object references, not pointers so that memory can be compacted.

Everything can be an Object

- Methods can work with objects so they can handle any type including primitive types (long, float).

```
void Serialize(Stream, object);
object Deserialize(Stream)
void ArrayList.Add(object)
```

- C++ or Java cannot use primitive types as objects.
- In Smalltalk, primitive types are objects, but using primitive types has a performance cost.
- In C# primitive types can be converted to objects when necessary.

Unified Type System

- Collections can be used with all types.
- Types are interoperable between .NET languages
  - Exceptions, Classes, Inheritance
- All types inherit from System.Object
- Object references avoid random pointer errors.
  - **cust**, **list** are object references
- Properties, Methods, Events, Interfaces, Delegates.
- Single Implementation Inheritance

Type Safety

- Code usually verified before compilation.
  - No buffer overwrites
  - Method entry and exit at well defined points.
  - No uninitialized variables
  - No unsafe casts
- Security Policy applies to type safe code.
- Type safe code prohibits pointer arithmetic to prevent subversion of the type system.
  - C# pointers are prohibited in type safe code.
- Allows for application domains.

Robust Software Development

- Garbage Collection – no memory leaks
- References – no random pointer overwrites
- Type Safety – code cannot be subverted
- Web pages can be written in C#

Interface-Based Programming

- Interfaces are a fundamental type.

```
public static void SaveFile(Stream s, IFormatter f, IList l) {
    f.Serialize(s, l);
    s.Close();
}
```

- Program to pure behavior, not implementation.
- With attributes and metadata, replace system functionality
  - ISerializable interface
- Multiple Interface Inheritance

Assemblies

- Programs are deployed as assemblies.
  - Assemblies are either executables or libraries.
    - Serialize.exe is an assembly
  - Metadata about types in assembly is stored with assembly (self-describing)
  - Assembly itself has metadata
    - Describes assemblies dependencies
    - Version of assembly

Assembly Metadata

```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 1:0:5000:0
}
.assembly extern System.Runtime.Serialization.Formatters.Soap
{
  .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A )
  .ver 1:0:5000:0
}
.assembly Serialize
{
  ...
 Ver 1:0:0:0
}
```

Assembly Version Policy

- Version is part of the assembly name.
  - Unique name based on public/private keys.
- Private deployment
  - Copy all files to application directory.
  - No need for versioning or unique names.
- Public deployment in Global Assembly Cache (GAC) requires strong name.
  - Link to specified versions in config file.
- No more “DLL Hell”.

Component Development

- An assembly is a component.
  - Unified type system with language interoperability
  - Properties, methods, and events exportable
  - Design and run time attributes
  - No COM infrastructure to implement.
- **As**, **Is** C# constructs allow interface query.
  - **As** operator converts one interface type to another
  - **Is** operator checks if interface is supported
- Metadata means no IDL or header files needed.
- C# Components be used from Web pages.

Interoperability

- C# code can interoperate with:
  - C++
  - Win32 APIs
  - COM components
  - Other .NET languages
  - XML and SOAP
- Easy learning curve from C++ or Java.

Type Declarations

- Value Types (contain data, cannot be null)
  - struct **struct Point {int x; int y}**
  - primitive types
  - enum **enum Answer : int {Yes, No}**
- Reference Types (refer to an object, can be null)
  - Class
  - Interface
  - Delegate
  - array (implements System.Array)
  - String (alias for System.String)
- Reference types are allocated on the heap. Value Types can be allocated on the stack, or on the heap if part of a reference type.

Type Members

- No Global Variables in C#
- Structs and Classes can have members:
  - Fields
  - Constants, ReadOnly
  - Methods
  - Properties
  - Indexers
  - Operators
  - Constructors
  - Finalizers (use C++ destructor notation)

Checking Account Example

```
public class CheckingAccount : Account
{
    public CheckingAccount()  { balance = 100;}
    public override void Deposit(decimal amount) {balance += amount;}
    public override void Withdraw(decimal amount)
    {
        balance -= amount;
        if (balance < 0) throw new Exception("Negative Balance.");
    }
    public void Show()
    {
        Console.WriteLine("balance = " + Balance);
    }
}
```

Primitive Types

• Signed	sbyte, short, int, long
• Unsigned	byte, ushort, uint, ulong
• Character	char
• Floating Point	double, float, decimal
• Boolean	bool

- Aliases for system types:
  - bool ⇨ System.Boolean

Class

- Single Implementation Inheritance
- Multiple Interface Inheritance
- Members can be static or instance
- Can have nested types
- Access can be public, private, protected or internal

Inheritance Intent

- To help solve the fragile basic class problem:
  - methods are marked **abstract** or **virtual**
  - they are not virtual by default
  - methods in derived classes are marked **new** or **override**

Boxing and Unboxing

- Value Types can be converted to Reference types when necessary

```
int x = 10;
object o = x;
string s = o.ToString();
int y = (int)o;
```

Delegate

- Type safe function pointers

```
public delegate int RegisterCustomer(string firstName, string LastName);
public void Process(RegisterCustomer customerFunc) {…}
```

- Each delegate has an invocation list with type safe methods for adding and removing from the list.

Events Use Delegates

```
public delegate void EventHandler(object sender, EventArgs e);

public class MenuItem
{
    public event EventHandler Click;

   protected void OnClick(EventArgs e) {
       if (Click != null) Click(this, e);}}
 }
 …
MenuItem menuItem1 = new MenuItem();
menuItem1.Click += new System.EventHandler(Draw_Click);
private void Draw_Click(object sender, System.EventArgs e) {…}
```

Properties

- Properties are methods treated as public fields.

```
private decimal balance;
public decimal Balance
{
    get { return balance;}
    set { caption = value; ComputeInterest();}
}
```

- Used just like a field

```
decimal amount = account.Balance;
```

Indexers

- Access object as if it was an array.

```
public class List
       ….
   private string[] names;
     public string this[int index]
    {
       get {return names[index];}
       set {names[index] = value;}
    }

 List list = new List();
 string first = list[2];
 list[1] = “John Doe”;
```

Improved C++ Expressions

- Conditionals must evaluate to a boolean.
- Switch statement has no automatic fall through.
- foreach loop (read-only)
- = is illegal in a conditional

C# Concepts are .NET Concepts

- NET is a virtual execution environment
  - Defined in ECMA-335.
  - ECMA-334 is the C# specification
- Program to a logical model.
  - Compilers produce intermediate code, not native code.
- Logical to physical translation to physical code happens on users machine through JIT compilation, not on the developer's machine.

Logical Programming Model

- The Common Language Runtime (CLR)
    - Memory management
    - Security
- The Common Type System (CTS)
    - Unified Type System
    - Extensible metadata
- The Common Intermediate Language (CIL)
    - Stack based, object
- The Common Language Specification (CLS)
    - Language Interoperability
- Framework Class Library (FCL)

Intermediate Language

- All .NET compilers emit Intermediate Language.
  - ILDASM (IL Disassembler) can be used to view the IL code and metadata. Useful for debugging and understanding system code.
- CTS and IL make it possible for languages to interoperate.
  - IL code can be verified for all platforms.
- CLS defines language interoperability.
  - Case sensitivity in public and protected members.
  - Allows FCL to be used by all languages.

Serialize.exe MSIL

```
IL_0000:  newobj     instance void [mscorlib]System.Collections.ArrayList::.ctor()
IL_0005:  stloc.0
IL_0006:  newobj     instance void Customer::.ctor()
IL_000b:  stloc.1
IL_000c:  ldloc.1
IL_000d:  ldstr      "Charles Darwin"
IL_0012:  stfld      string Customer::name
IL_0017:  ldloc.1
IL_0018:  ldc.i4.s   10
IL_001a:  conv.i8
IL_001b:  stfld      int64 Customer::id
IL_0020:  ldloc.0
IL_0021:  ldloc.1
IL_0022:  callvirt   instance int32
    [mscorlib]System.Collections.ArrayList::Add(object)
IL_0027:  pop
```

Managed vs. Type Safe Code

- Garbage Collection is one of the services provided by the Common Language Runtime .
  - Data under CLR garbage collection control is managed data.
  - Code using CLR features is managed code.
- Managed code is not automatically type safe.
  - C++

Summary

- C# is a programming language that is a streamlined version of C++ with less complexity.
- Memory references and garbage collection remove major impediments to producing quality code.
- Since all types can be treated as objects, the programming model is more powerful.
- Components can be easily developed.
- Development is faster.