

Understanding Assemblies and Reflecting on them

Martin Kropp
University of Applied Sciences Northwestern Switzerland

Learning Target

• You

- can explain the concept and format of .Net assemblies
- can explain the content of a .Net assembly
- can explain the concept of the global assembly cache
- can develop a public assembly
- understand and can explain the .Net Reflection concept
- can apply the .Net reflection API to analyze assemblies
- can apply the .Net reflection API to dynamically load assemblies and classes

Content

Assemblies

- Role and Format of .Net Assemblies
- The Global Assembly Cache
- Managing Private and Public Assemlies

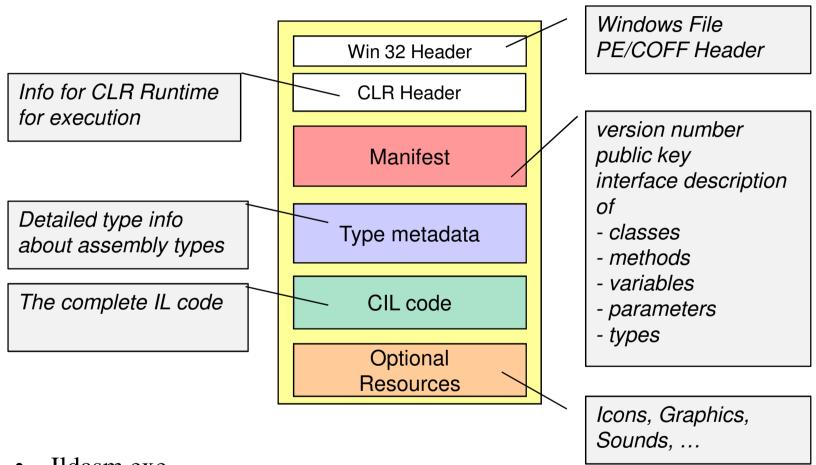
Reflection

- Type Information in .Net Assemblies
- Loading and Analyzing .Net Assembly

Role of .Net Assemblies

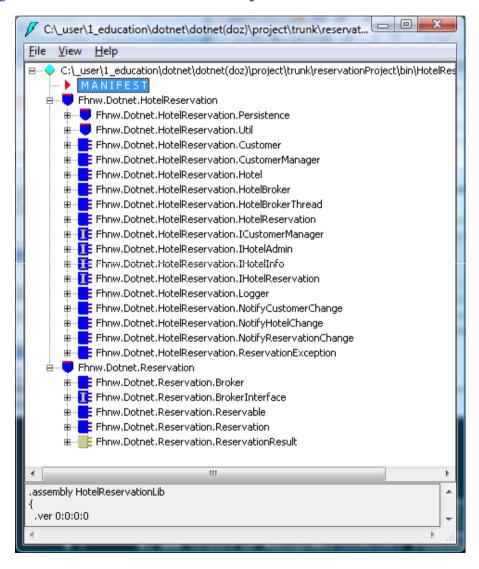
- Assemblies Promote Code Reuse
 - Are the smallest executable unit
- Assemblies Establish a Type Boundary
 - Types in a .Net assembly are considered to be unique
- Assemblies Are Versionable Units
 - Contain exact version info
- Assemblies Are Self-Describing
 - Contain all type info
- Assemblies Are Configurable
 - Can be stored anywhere

The Format of a .Net Assembly



- Ildasm.exe
- dumpbin.exe [/headers | /clrheader]

Format of .Net Assembly – ILDASM - View



Assembly Manifest

```
_ D X
MANIFEST
                                                                 External References
Find Find Next
// Metadata version: v2.0.50727
.assemblu extern mscorlib
  .publickeutoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 2:0:0:0
                                                                 Assembly Information
.assembly extern System.Data.SQLite
  .publickeytoken = (DB 93 7B C2 D4 4F F1 39 )
  .ver 1:0:58:0
                                                                                             - - X
                 MANIFEST
                 Find Find Next
.assembly exterr
                  .assembly HotelReservationLib
  .publickeytoke {
  .ver 2:0:0:0
                   // --- The following custom attribute is added automatically, do not uncomment -----
                   // .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute::.ctor(val
.assembly extern
                    .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationRelaxation
  .publickeytoke
                    .custom instance void [mscorlib]System.Runtime.CompilerServices.RuntimeCompatibilityA
  .ver 2:0:0:0
                    .hash algorithm 0x00008004
                    .ver 0:0:0:0
      -111
                  .module HotelReservationLib.dll
                 // MUID: {8A77290D-2E32-4D4A-843A-CFD9216F3BC6}
                  .imaqebase 0x00400000
                  .file alignment 0x00001000
                  .stackreserve 0x00100000
                  .subsystem 0x0003
                                         // WINDOWS CUI
                  .corflags 0x00000001
                                       // ILONLY
                 // Image base: 0x00A40000
```

... and the IL Code

HotelBroker.AddHotel

```
Fhnw.Dotnet.HotelReservation.HotelBroker::AddHotel : int64(string,string,int32,valuetype [ms... 🖵 📮
Find Find Next
.method public hidebysiq newslot virtual final
        instance int64 AddHotel(string city,
                                  string name,
                                  int32 number,
                                  valuetupe [mscorlib]System.Decimal cost) c
                     98 (0x62)
 // Code size
  .maxstack 6
  .locals init ([0] class Fhnw.Dotnet.HotelReservation.Hotel hotel,
           [1] int64 CS$1$0000,
           [2] bool CS$4$0001)
 IL 0000: nop
 IL 0001: 1darq.0
 IL 0002: 1darq.1
 IL 0003: 1darq.2
 IL 0004: call
                       instance int64 Fhnw.Dotnet.HotelReservation.HotelBro
 IL 0009: 1dc.i4.m1
 IL_000a: conv.i8
  IL 000b: ceq
  IL 000d: stloc.2
  II RARA - 1dlac 2
```

Managing Private Assemblies

• Usually located in the executable's directory

Net Probing Process

- The process how the runtime resolves the location of private external assemblies
- 1. Look for DLL in clients assembly location
- 2. Look for EXE in clients assembly location
- 3. Look for either DLL or EXE in a subdirectory with same name as the assembly in clients assembly location
- If not found "FileNot Found Exception"
- Usually stored in the local directory
- Can also be stored in any other location using the applications config file

Configuring Private Assemblies

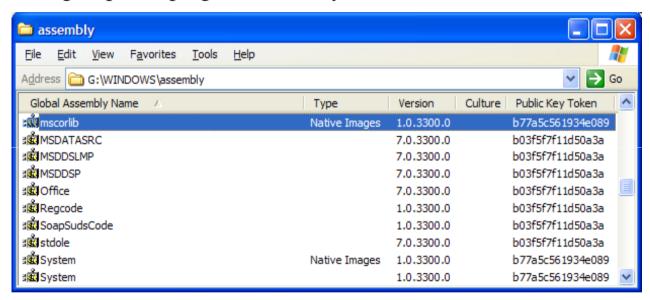
• Can also be stored in any other location using the applications config file (*AppName.exe.config*)

Managing Shared Assemblies

- Share the Assembly with other applications
 - Can be used several applications
 - Only one copy stored
- The Global Assembly Cache (GAC)
 - A safe Win\System32 repository ©
 - Can store different versions of the same assembly
 - Only limited and controlled access
- Various GACs
 - GAC .net 1.x
 - GAC_32 / GAC_64
 - GAC_MSIL

The Global Assembly Cache

- A View on the GAC
 - (using Explorer plugin "Assembly Cache Viewer" (shfushion.dll)



• The Actual path

C:\Windows\assembly\NativeImages1_v1.1.4322\mscorlib\1.0.5000.0__b77a5c561934e089_1dc9747c\mscorlib.dll

Installing Assemblies into the GAC

- No direct Access
- Installing with the GACUtil (gacutil.exe)
 - > gacutil -i GlobalLib.dll
- Removing assemblies from the GAC
 - > gacutil -u GlobalLib
- Only **signed** assemblies can be stored
 - Assembly is signed with a strong name

Strong Name for Assemblies

- "strong" assembly name consists of:
 - name
 - version number (major.minor.build.revision)
 - culture attribute (usually neutral)
 - hash value of a public key

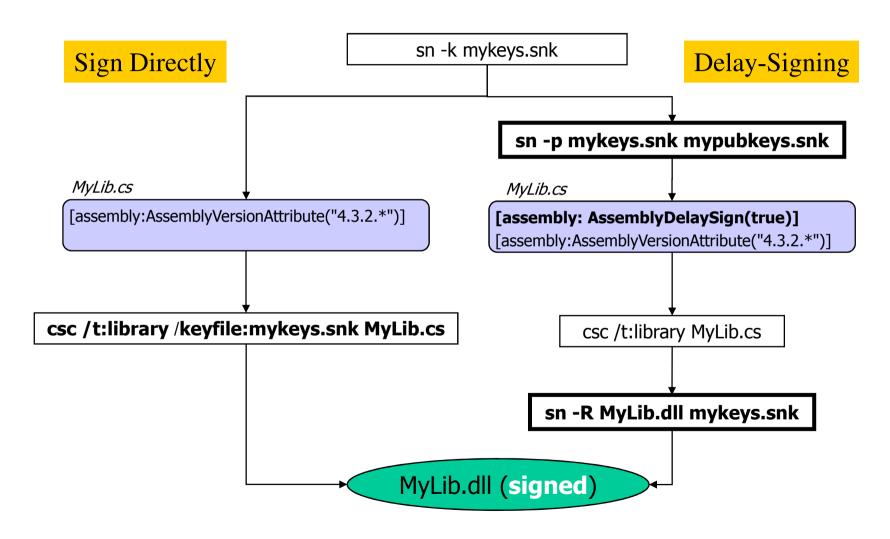
```
e.g. MyLib,
Version=1.2.745.18000,
Culture=en-US,
PublicKeyToken=13a3f300fff94cef
```

Strong Name Tool (sn.exe)

Command line tool for

- generating pairs of private & public keys
 sn -k mykeys.snk
- extracting the public key from a key pair
 sn -p mykeys.snk mypubkey.snk
- delayed signing of assemblies
 sn -R MyLib.dll mykeys.snk

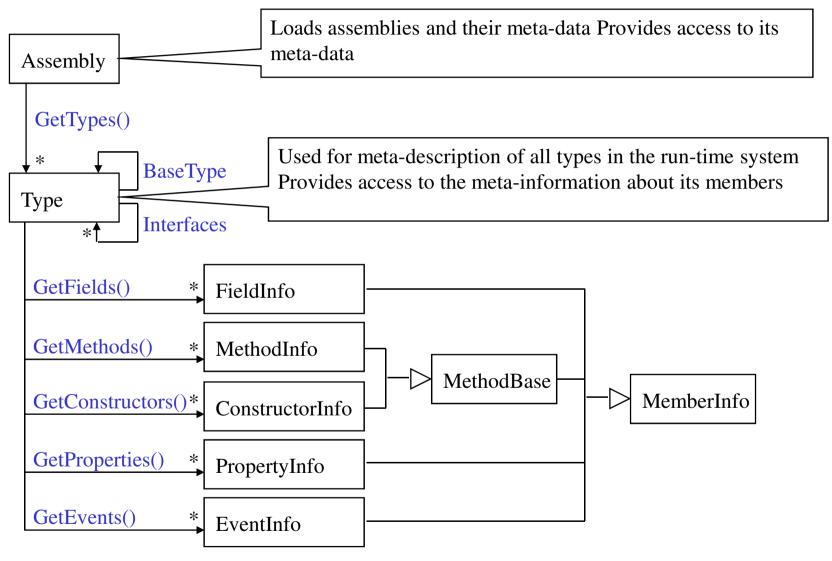
Signing an Assembly



Reflection

- Permits access to meta-information of types at run-time
- System.Reflection allows:
 - Getting meta-information about assemblies, modules and types
 - Getting meta-information about the members of a type
 - Dynamic creation of instances of a type at run-time
 - Search for methods and their dynamic invocation at run-time
 - Accessing values of properties and fields of an object
 - Design of new types at run time
 - → namespace *System.Reflection.Emit*

Reflection Class Hierarchy



Example Reflection

• Loading the assembly "HelloWorld.exe":

```
Assembly a = Assembly.Load("HelloWorld");
```

• Print all existing types in a given assembly

```
Type[] types = a.GetTypes();

foreach (Type t in types)

Console.WriteLine(t.FullName);

Hello.HelloWorld
```

namespace Hello {

public class HelloWorld {

public static void Main(string[] args)

public override string ToString() {

• Print all existing methods of a given type

```
Type hw = a.GetType("Hello.HelloWorld");
MethodInfo[] methods = hw.GetMethods();
foreach (MethodInfo m in methods)
Console.WriteLine(m.Name);

Main
ToString
```

Example Reflection (2)

• Create a new instance of a given type

```
Assembly a = Assembly.Load("HelloWorld");
object o = a.CreateInstance("Hello.HelloWorld");
```

- Get method ToString(), which has no parameters
- Invoke the method

```
Type hw = a.GetType("Hello.HelloWorld");
MethodInfo mi = hw.GetMethod("ToString");
object retVal = mi.Invoke(o, null);
```

Reflection.Emit

- Reflection.Emit allows creation of assemblies and types at run-time
 - creation of assemblies
 - creation of new modules
 - creation of new types
 - creation of symbolic meta-information of existing modules
- System.Reflection.Emit supports realization of .NET compilers und interpreters
- Important classes of Reflection.Emit are

AssemblyBuilder to define assemblies

ModuleBuilder to define modules

TypeBuilder to define types

MethodBuilder to define methods

ILGenerator to emit IL-code