

Practical Reflection

Using reflection in .NET while still keeping your sanity

Jeremy Clark

www.jeremybytes.com

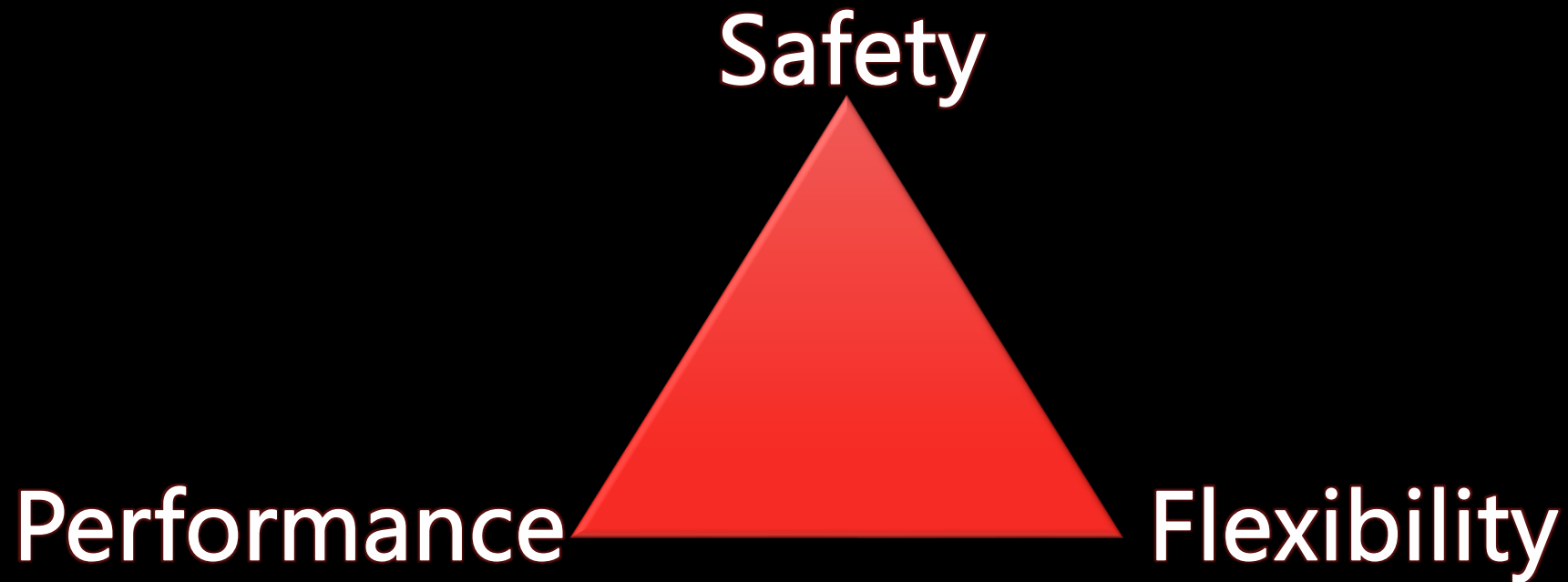
@jeremybytes

Just for
Experts?



Goal

Explore the Practical Parts of Reflection



What is Reflection?

Inspecting the metadata and compiled code in an assembly.

- What is an assembly?
- What is metadata?
- How is the code compiled?

.NET Assemblies

Assembly
(exe or dll)

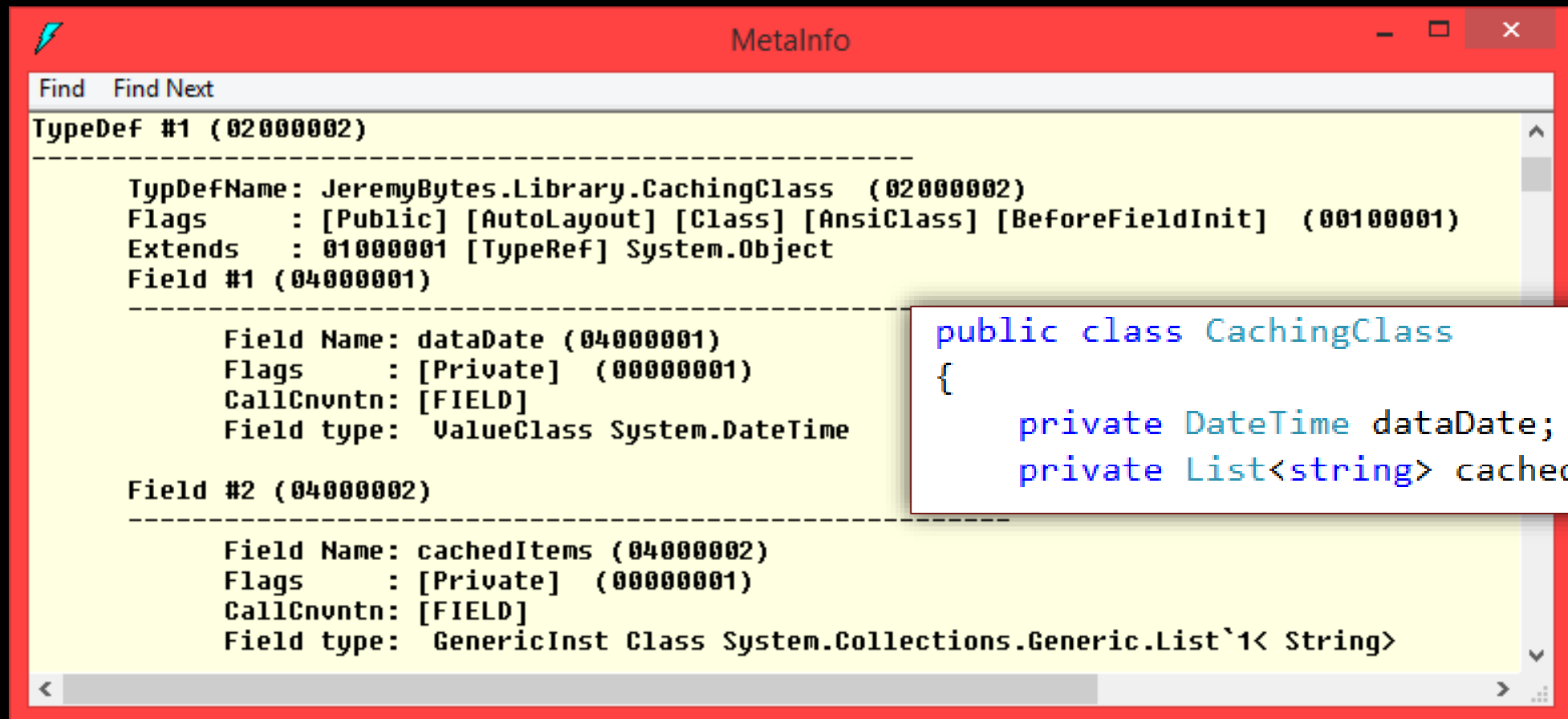
Module

*Assembly
Manifest*

*Metadata
+ IL*

*Resources
(optional)*

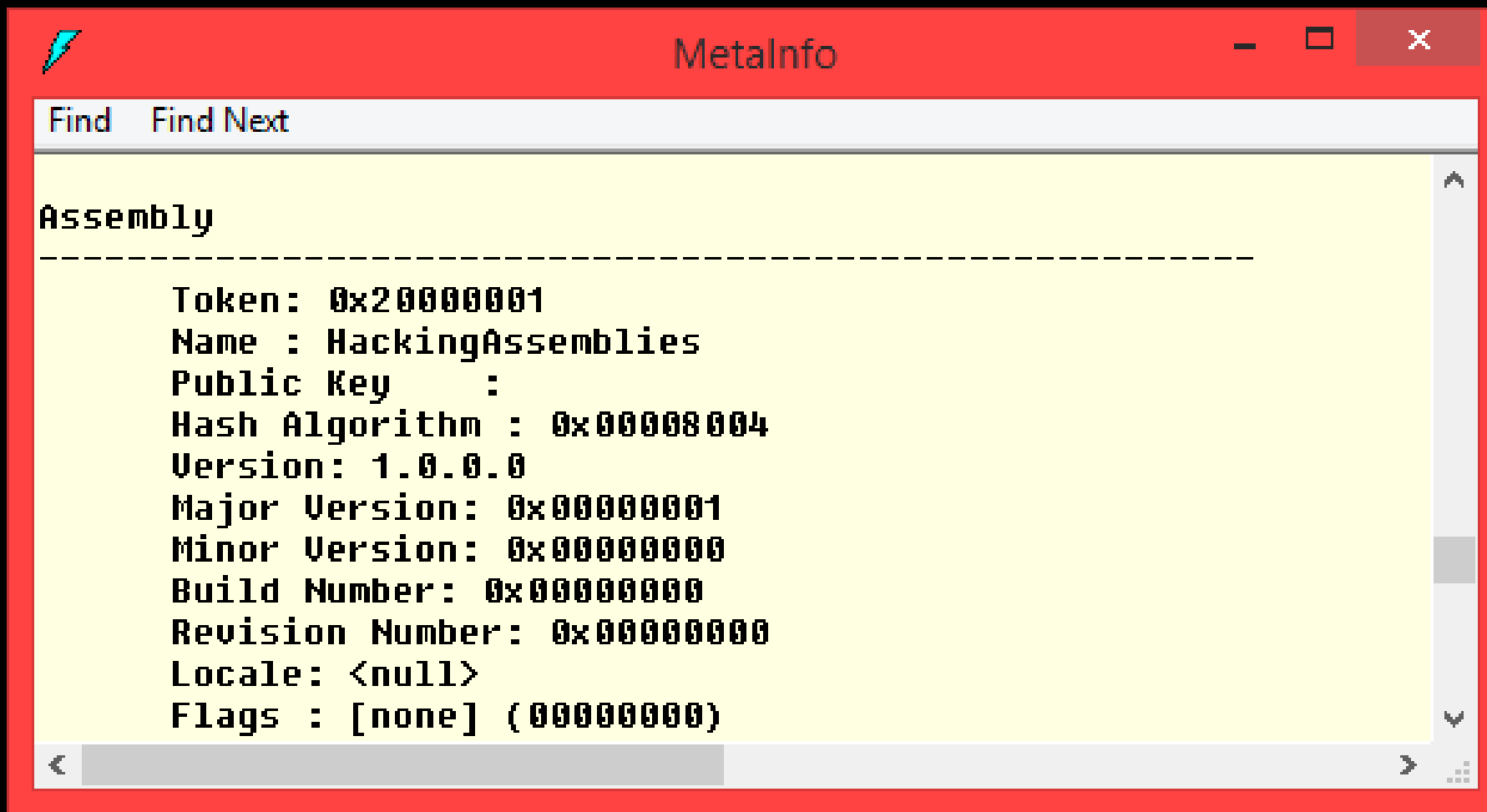
Type Definitions



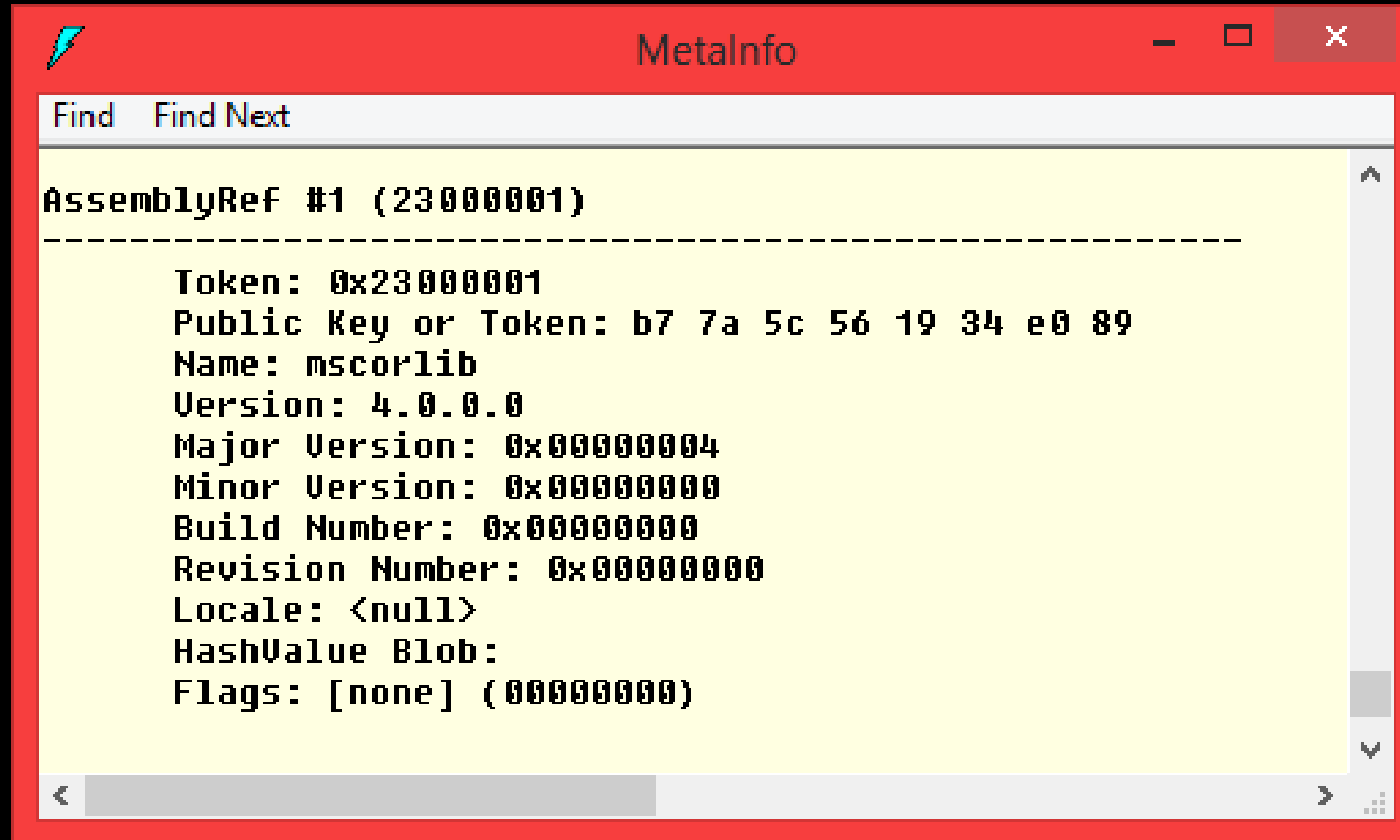
The screenshot shows a window titled "MetaInfo" with a search bar at the top containing "Find" and "Find Next". Below the search bar, the text "TypeDef #1 (02000002)" is displayed. A dashed line separates this header from the details. The details include: "TypeDefName: JeremyBytes.Library.CachingClass (02000002)", "Flags : [Public] [AutoLayout] [Class] [AnsiClass] [BeforeFieldInit] (00100001)", "Extends : 01000001 [TypeRef] System.Object", and "Field #1 (04000001)". Another dashed line follows. Below it, "Field Name: dataDate (04000001)" is shown, followed by "Flags : [Private] (00000001)", "CallCnvn: [FIELD]", and "Field type: ValueClass System.DateTime". A third dashed line follows. Below it, "Field #2 (04000002)" is shown, followed by "Field Name: cachedItems (04000002)", "Flags : [Private] (00000001)", "CallCnvn: [FIELD]", and "Field type: GenericInst Class System.Collections.Generic.List`1< String>". A scroll bar is visible on the right side of the text area.

```
public class CachingClass
{
    private DateTime dataDate;
    private List<string> cachedItems;
```


Assembly Information

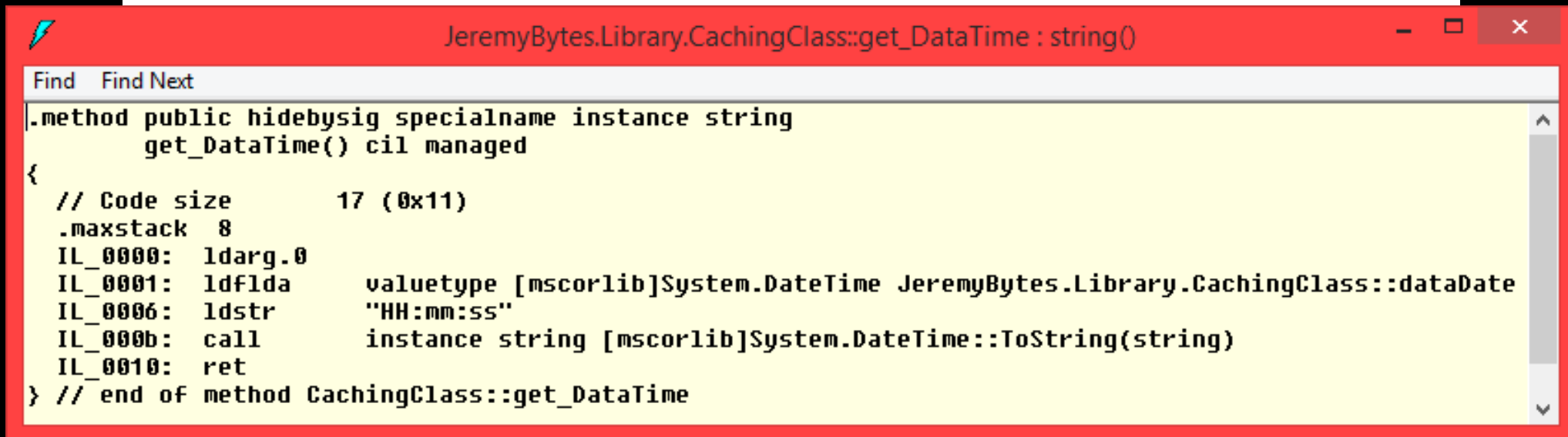


Referenced Assemblies



IL (Intermediate Language)

```
public string DateTime
{
    get { return dataDate.ToString("HH:mm:ss"); }
}
```



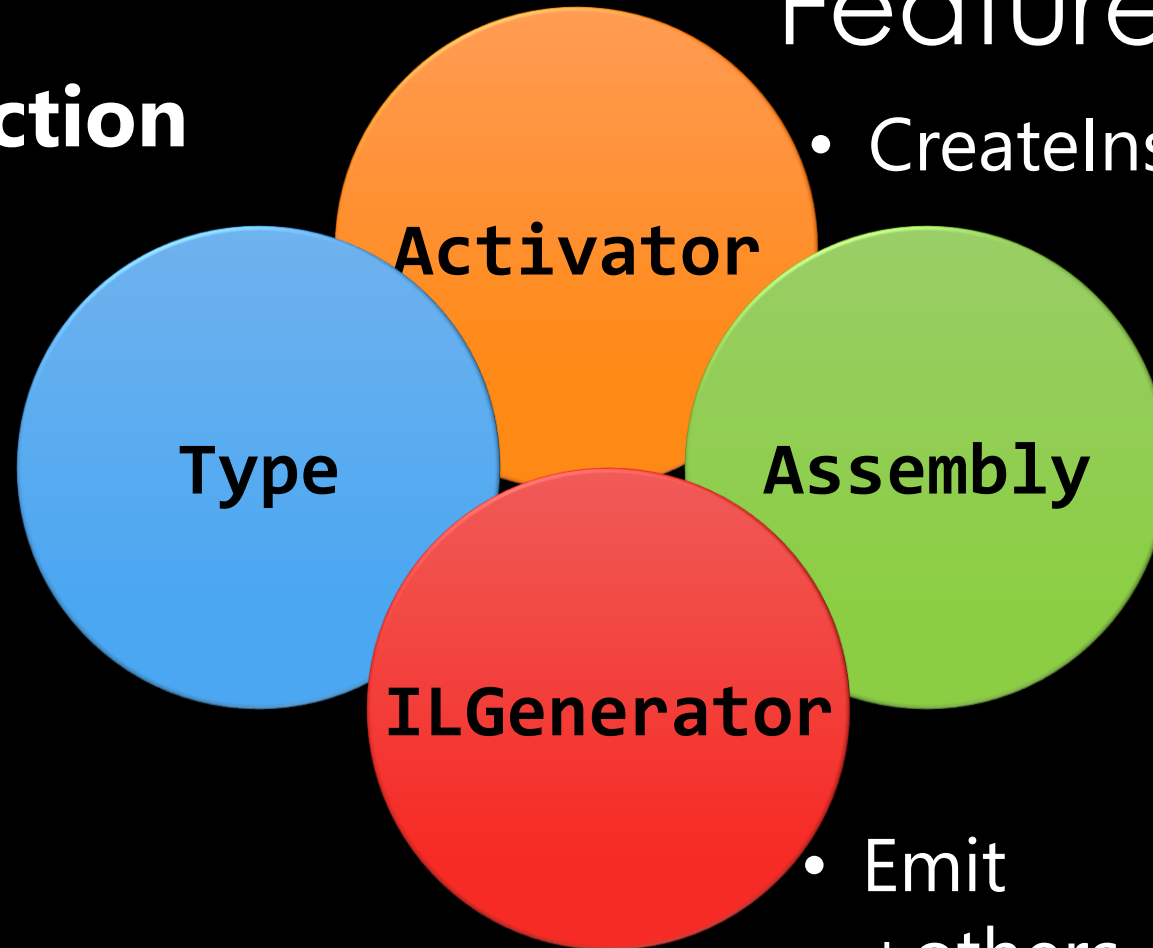
The screenshot shows a Visual Studio window with the title bar "JeremyBytes.Library.CachingClass::get_DataTime : string()". The window contains a code editor with the following IL code:

```
Find Find Next
.method public hidebysig specialname instance string
    get_DataTime() cil managed
{
    // Code size          17 (0x11)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: ldflda      valuetype [mscorlib]System.DateTime JeremyBytes.Library.CachingClass::dataDate
    IL_0006: ldstr        "HH:mm:ss"
    IL_000b: call       instance string [mscorlib]System.DateTime::ToString(string)
    IL_0010: ret
} // end of method CachingClass::get_DataTime
```

System.Reflection

Feature Overview

- GetType
- GetMember
- GetMethod
- GetProperty
- GetField
- + many more



- CreateInstance

- Emit
- + others

- Load
- LoadFrom
- GetTypes
- GetName
- GetFiles
- + many more

Things You Can Do

- Reflecting on a Property

```
Type sampleType = typeof(SampleClass);  
PropertyInfo cacheProperty = sampleType.GetProperty("CachedItems");  
List<string> cacheValue = cacheProperty.GetValue(privateSample)  
    as List<string>;
```

- Useful for interacting with COM objects (pre-.NET 4.0)
- “dynamic” is a better choice for interacting with COM

Things You Can Do

- Reflecting on a Method

```
var list = new List<int>();  
Type listType = typeof(List<int>);  
Type[] parameterTypes = { typeof(int) };  
MethodInfo addMethod = listType.GetMethod("Add", parameterTypes);  
addMethod.Invoke(list, new object[] { 7 });
```

- Useful for interacting with COM objects (pre-.NET 4.0)
- “dynamic” is a better choice for interacting with COM

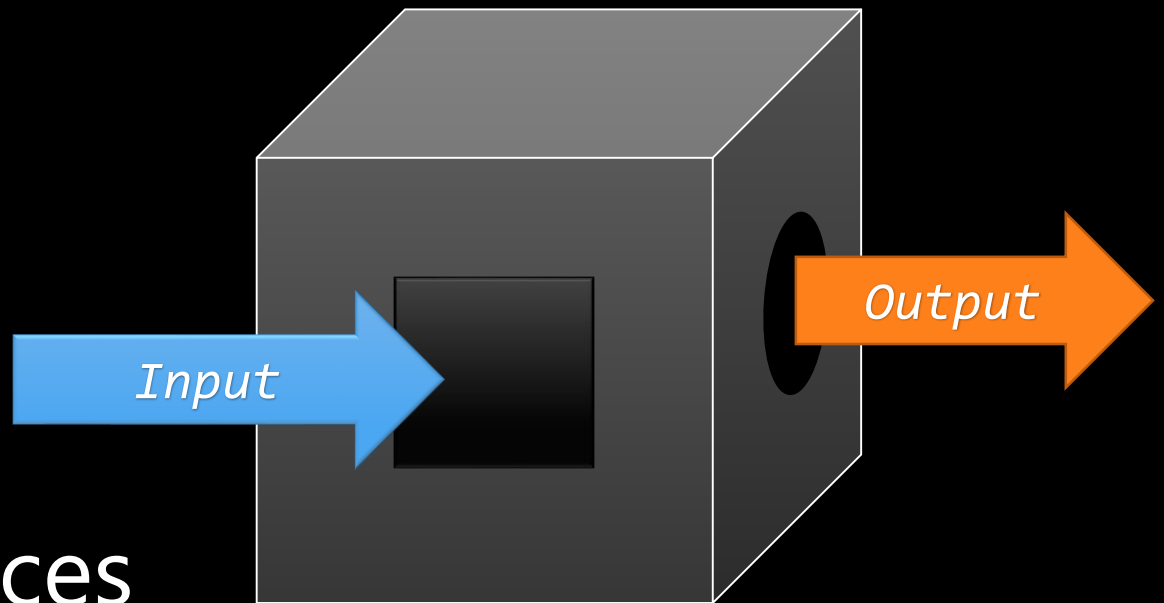
Things You Can Do

- Reflecting on a Private Field

```
Type sampleType = typeof(SampleClass);  
FieldInfo cacheField = sampleType.GetField("internalCache",  
                                           BindingFlags.NonPublic | BindingFlags.Instance);  
List<string> cacheValue = cacheField.GetValue(privateSample)  
    as List<string>;
```

- BindingFlags give us access to non-public members
- DANGER DANGER DANGER

Encapsulation



- Use the exposed interfaces
- Don't peek inside the box

Demo

Performance Concerns



Best Practice

Program to an abstraction
rather than a concrete type

Practical Reflection Strategy

- **Dynamically Load Assemblies**
 - Happens one time (at start up)
- **Dynamically Load Types**
 - Happens one time (at start up)
- **Cast Types to a Known Interface**
 - All method calls go through the interface
 - No dynamic method calls – no `MethodInfo.Invoke`
 - Avoid interacting with private members

Various Data Sources

Microsoft SQL Server

MongoDB

CSV

REST Service

Oracle

WebAPI

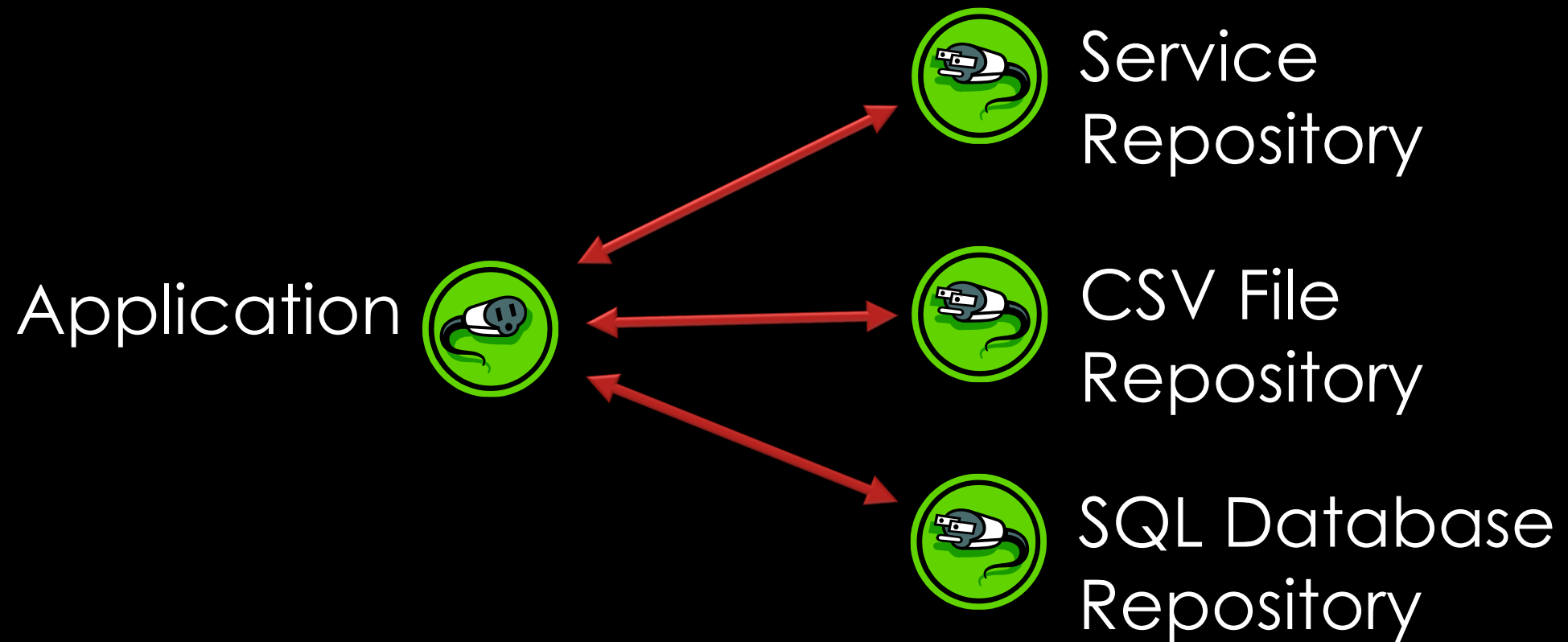
Amazon AWS

JSON

Hadoop

Microsoft Azure

Pluggable Repositories



Demo

Run-Time Binding

The background of the slide features abstract, flowing, translucent waves. On the left side, there are red waves, and on the right side, there are blue waves. These waves overlap and flow across the bottom of the slide, creating a dynamic, fluid-like effect against the solid black background.

Benefits of Dynamic Loading

- Only ship 1 repository assembly
- Remove dependency on concrete repositories
- New repositories can be added without modifying exiting code

Assembly-Qualified Type Name

```
PersonRepository.SQL.SQLRepository,  
PersonRepository.SQL,  
Version=1.0.0.0,  
Culture=neutral,  
PublicKeyToken=b77a5c561934e089
```

- Fully-qualified type name (namespace and type)
- Assembly Name
- Assembly Version
- Assembly Culture
- Assembly Public Key (for strongly-named assemblies)

Limiting Reflection

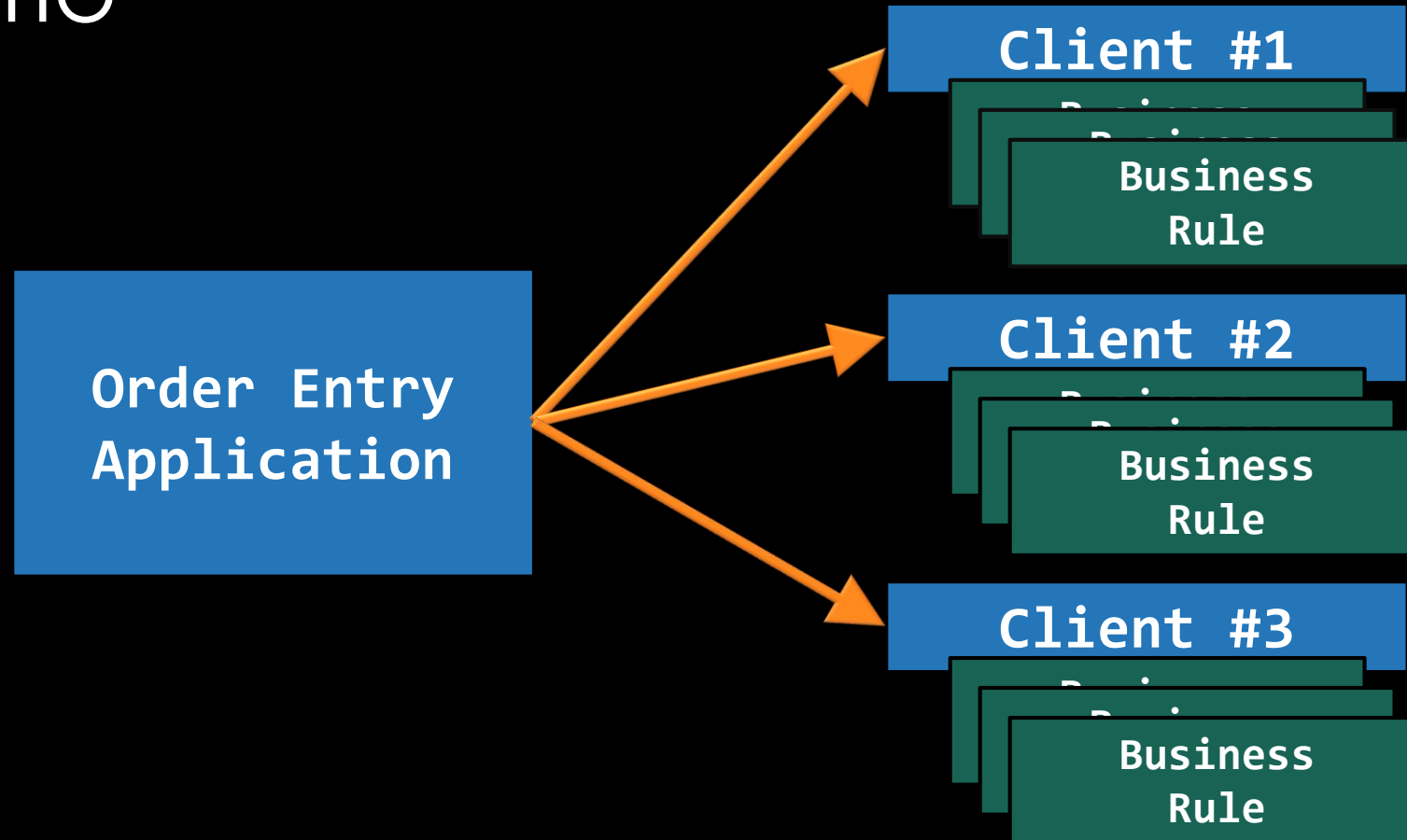
```
private void FetchButton_Click(object sender, EventArgs e)
{
    ClearListBox();

    var people = repository.GetPeople();
    foreach (var person in people)
        PersonListBox.Items.Add(person);

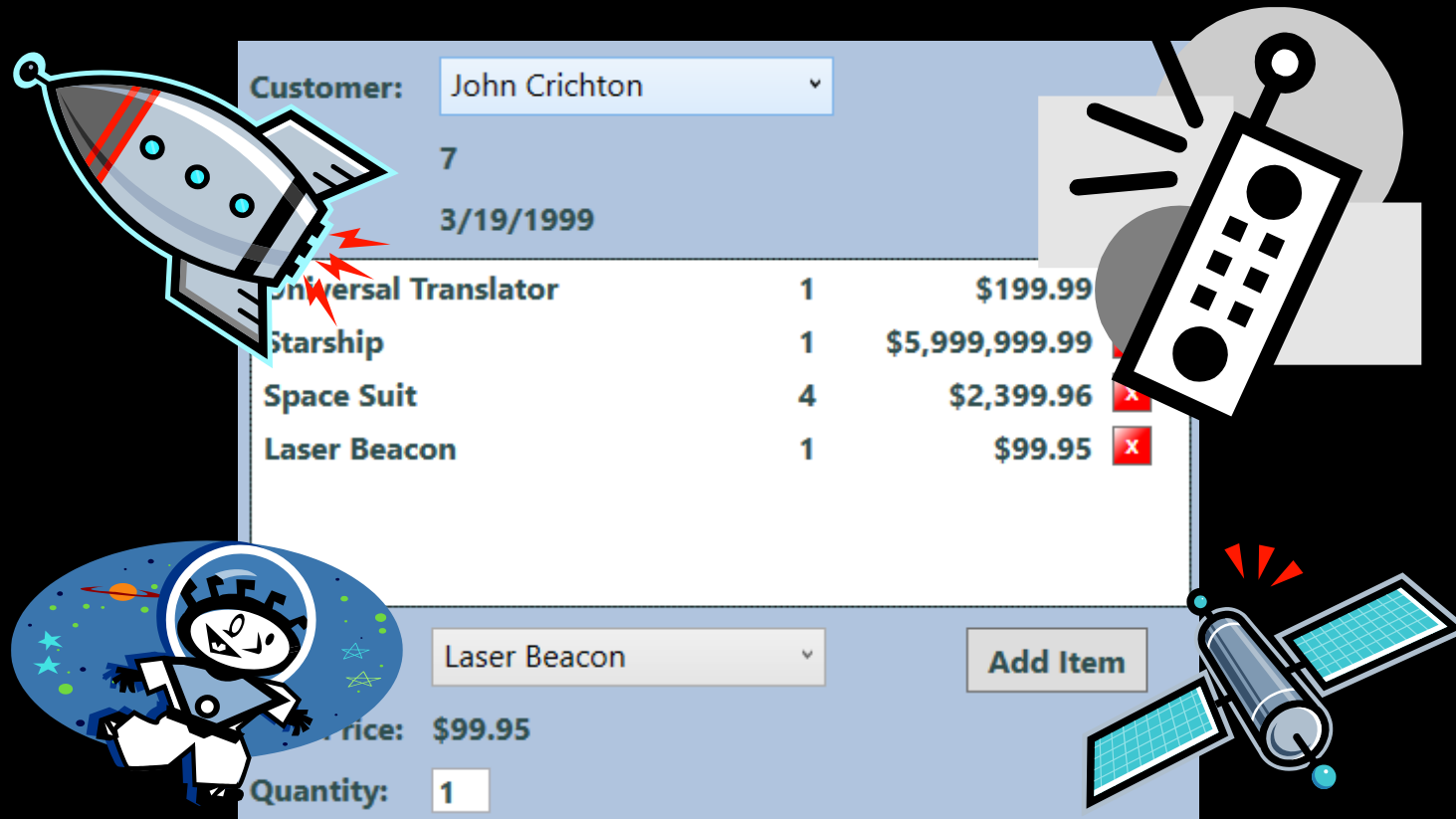
    ShowRepositoryType(repository);
}
```

- No Reflection Here
- Method calls through `IPersonRepository`

Scenario



Application



Customer: John Crichton

7

3/19/1999

Universal Translator	1	\$199.99	
Starship	1	\$5,999,999.99	
Space Suit	4	\$2,399.96	x
Laser Beacon	1	\$99.95	x

Laser Beacon

Add Item

Price: \$99.95

Quantity: 1

The application interface is decorated with several space-themed illustrations: a rocket ship with red lightning bolts on the left; a satellite with a circular antenna on the top right; a cartoon astronaut floating in a blue oval with stars on the bottom left; and a satellite with solar panels on the bottom right.

Business Rule Interface

```
public interface IOrderRule
{
    string RuleName { get; }
    OrderRuleResult CheckRule(Order order);
}

public class OrderRuleResult
{
    public bool Result { get; set; }
    public string Message { get; set; }

    public OrderRuleResult(bool result,
                           string message) {...}
}
```

Business Rules

Maximum
Discount based
on
Customer
Rating

Only 1
Captain's
Chair
Allowed

Maximum of
1 Starship
per Order

Name Badge
must match
Customer Name

Discovery Process

- Locate all assemblies in the “Rules” folder
- Load each assembly
- Enumerate the types in the assembly
- Check each type to see if it implements our Rule interface
- Create an instance of each Rule and add it to the Rule Catalog



Thank You!

Jeremy Clark

- <http://www.jeremybytes.com>
- jeremy@jeremybytes.com
- @jeremybytes